MSX

PROGRAMMEREN IN MACHINETAAL



```
                  ;Bewegen v.d. sprites

                  ;Vertraag
E05B 010002       VRTRG:        LD    BC,0200H
E05B CDB700       VRTRG1:       CALL  BREAKX
E05E 383D                       JR    C,RSTSCR
                  ;Bepaal of de sprites
                  ;elkaar raken
E060 CD3E01                     CALL  RDVDP
E063 CB6F                       BIT   5,A
E065 2807                       JR    Z,VRTRG2
                  ;Maak sprite 1 onzichtbaar
E067 AF                         XOR   A
E06B 21071B                     LD    HL,1B07H
E06B CD4D00                     CALL  WRTVRM
E06E 0B           VRTRG2:       DEC   BC
E06F 79                         LD    A,C
E070 B0                         OR    B
E071 20EB                       JR    NZ,VRTRG1
                  ;Verplaats sprite 0
```

Volledige cursus programmeren in machinetaal. Alles over de Z 80 processor en het interne leven van de MSX-computer.

M. B. IMMERZEEL

DE MUIDERKRING

MSX

PROGRAMMEREN
IN MACHINETAAL

ISBN 90 6082 260 9

M.B.IMMERZEEL

MSX

PROGRAMMEREN IN MACHINETAAL

DE MUIDERKRING B.V. - WEESP
UITGEVERIJ VAN TECHNISCHE BOEKEN EN TIJDSCHRIFTEN

Inhoud

Deel 1. De computer algemeen.

Deel 2. De MSX-computer.

Voorwoord

Het is gebruikelijk dat iedereen, die een hobbycomputer koopt, direct zijn gang kan gaan met het programmeren in de BASIC- programmataal. Dat is niet altijd zo geweest. De eerste hobbycomputers hadden vaak geen andere mogelijkheid dan het invoeren van hexadecimale codegetallen waarmee de processor opdrachten konden worden gegeven. Dat is het programmeren in "machinetaal", de enige taal die de processor kan verstaan. Al snel bleek dat het programmeren in machinetaal op zijn minst een geduldwerkje was en dat een "hogere" taal, zoals BASIC, het voordeel had in één opdracht gelijk al een groot aantal machinetaalinstructies in de computer te voeren. Want welke programmeertaal u ook gebruikt, altijd moeten de opdrachten worden omgezet, "vertaald", in een aantal instructies die de processor kent, machinetaalinstructies dus. Nu wordt er bij de hobbycomputers in het algemeen een "BASIC-interpreter" toegepast. Dat wil zeggen dat een BASIC-programma nooit in zijn geheel wordt omgezet in een machinetaal programma, maar dat regel voor regel steeds opnieuw weer wordt vertaald. U voert dan altijd weer een BASIC-programma in de computer, of het nu vanaf het toetsenbord is of vanaf de tape en steeds moet weer hetzelfde gebeuren, het vertalen. Nu kan dat in principe geen kwaad, als het programma's betreft waarvoor snelheid geen eerste vereiste is. Anders wordt dat als er sprake is van bewegingen over het scherm, bij bewegende beelden. Dan is snelheid altijd een voorwaarde voor het goed functioneren van het programma en die snelheid kan door het steeds opnieuw weer vertalen niet worden opgebracht. In die gevallen is machinetaalprogrammeren een noodzaak. Nu is er overigens wel een mogelijkheid om een volledig programma dat in een hogere taal is geschreven, volledig om te zetten in een machinetaalprogramma. De programma's die hiervoor moeten worden gebruikt, de "compilers" vragen echter een zeer grote geheugenruimte en dat is iets waar de hobbycomputers niet altijd zo ruim inzitten. Daarbij komt dat een machinetaalprogramma dat op die manier ontstaat, veel meer instructies bevat dan wanneer het direct in machinetaal was geschreven. De computerhobbyist die meer wil met zijn computer is daarom voor bepaalde programmagedeelten wel aangewezen op de machinetaal.

Machinetaalprogrammeren moet echter ook worden geleerd. De bedoeling is dat dit boek u daarbij tot een goede steun is. Noodzakelijk is dat u in elk geval de algemene opbouw van een computer kent, want daarop is de werking van de processor gericht. Ook is het nodig enig inzicht te hebben in de inwendige organisatie van de processor, zodat u de functies van de diverse registers duidelijk zijn. U kunt dan ook beter de werking van de diverse adresseermethoden begrijpen. Om in machinetaal te kunnen programmeren is het niet alleen nodig deze adresseermethoden te kennen, maar u moet ook inzicht hebben in de werking van de diverse instructies. Dit zijn allemaal zaken die uitsluitend betrekking hebben op de gebruikte processor en algemeen zijn. Daarom is het eerste gedeelte van dit boek niet strikt gericht op de MSX-computer. Het geeft de basiskennis die voor elke computer, die met de Z-80 processor werkt, een noodzaak is. Overigens is deze Z-80 een processor die in veel professionele computers is toegepast en ook zijn opvolgers heeft in de meeste populaire 16- bits personal computers.

Het tweede gedeelte van het boek handelt in elk geval over de mogelijkheden die u heeft bij het machinetaalprogrammeren met uw MSX-computer. Voor het leren programmeren in machinetaal is het noodzakelijk dat u de oplossingen kent van een aantal problemen die in diverse programma's kunnen voorkomen. Ook is het belangrijk om een aantal machinetaalprogramma's te bestuderen. U krijgt daarom een aantal programma's voorgeschoteld die eenvoudig en klein beginnen en later wat uitgebreider zijn. Al die programma's hebben een bepaalde werking en kunnen in uw computer worden uitgeprobeerd. Voor elk programma is uiteraard een volledige beschrijving van de werking gegeven. Dat deze programma's ook zijn gericht op de gebruiksmogelijkheden van uw computer is een voorwaarde voor dit boek geweest. Er zijn in

uw computer al een aantal machinetaalprogramma's aanwezig die door de programmeur kunnen worden gebruikt. Op welke manier de voornaamste van deze routines in uw programma's kunnen worden opgenomen wordt dan ook gedemonstreerd. Uiteindelijk is het de bedoeling dat u aan het eind van dit boek in staat bent om zelf uw machinetaalprogramma's samen te stellen, eventueel met gebruikmaking van een aantal van de in dit boek gegeven routines.

Ede, najaar '86

DEEL 1. DE COMPUTER ALGEMEEN

1. De opbouw van de computer

1.1. De samenstellende delen.

De meesten die zich gaan wagen aan het programmeren in machinetaal zullen al op de hoogte zijn met het programmeren in de BASIC programmataal. Uiteindelijk is dat de programmataal waarmee zij direct bij het voor de eerste keer inschakelen van de computer geconfronteerd werden. Ik neem dan ook aan dat u al enige ervaring hebt met het programmeren in BASIC. U bent dan al op de hoogte van het bestaan van een geheugen in uw computer en van een processor die alle opdrachten, die hem door het programma worden gegeven, uitvoert. Deze processor is het meest centrale onderdeel in de computer. Er bestaan diverse processoren en in uw computer bevindt zich de Z80 processor. Deze is ook in veel professionele computers toegepast.

Heeft u een BASIC programma in uw computer gevoerd, dan kan de processor de opdrachten die in dat programma worden gegeven, niet rechtstreeks uitvoeren. De opdrachten kunnen hem slechts worden toegediend in de vorm van codegetallen terwijl de soort van opdrachten die hij kan uitvoeren geheel verschillend is van de BASIC statements.

Dat wil zeggen dat een BASIC programma moet worden omgezet (vertaald) in een programma dat geschikt is voor de processor. Dit vertalen moet in de computer gebeuren door de processor zelf, zodat hiervoor een "vertaalprogramma" nodig is. Dit programma wordt de "BASIC interpreter" genoemd. Uiteraard is de BASIC interpreter een programma dat de juiste vorm heeft voor de processor. Een dergelijk programma heet een *machinetaalprogramma.* Een in machinetaal geschreven programma is zeer veel sneller dan een BASIC programma omdat dan het vertalen achterwege kan blijven.

Een programma en ook de gegevens die voor dat programma nodig zijn, worden in de computer opgeslagen in het geheugen. Een geheugen bestaat uit registers die binaire getallen kunnen bevatten van acht bits. Elk register is genummerd en zo'n nummer wordt een adres genoemd. Het laagste adres is 0, het volgende is 1 enzovoorts. De processor kan 65536 adressen de baas, van 0 tot en met 65535. Een geheugenregister is een elektronische schakeling en voor het "aanwijzen" van een geheugenplaats, *het adresseren*, zijn evenveel geleiders nodig als er bits nodig zijn voor het hoogste adres. Ook het adresseren geschiedt namelijk met binaire getallen. Er zijn daarom hiervoor zestien geleiders nodig ($2^{16} = 65536$). Deze zestien geleiders tesamen worden de *adresbus* genoemd. Voor het uitwisselen van de gegevens tussen het geadresseerde register en de processor zijn *acht* geleiders nodig, de *databus.*

Voor het inbrengen van de gegevens door middel van het toetsenbord, de cassetterecorder of de floppy disk zijn *ingangspoorten* nodig. Hierop zijn de geleiders aangesloten die de verbinding vormen met het desbetreffende randapparaat (toetsenbord en dergelijke) en de computer. Deze ingangspoorten zijn elektronische schakelingen die als registers zijn te beschouwen. Is de verbindingslijn van het randapparaat spanningsvoerend (+5V), dan is de ingangspoort "1", anders "0".

De ingangspoorten zijn samengevoegd tot registers van acht bits, zodat op een "poortregister" acht ingangslijnen kunnen worden aangesloten.

Voor het uitvoeren van gegevens naar de randapparatuur (beeldscherm, floppy disk, cassetterecorder en printer) zijn eveneens poortschakelingen nodig, de *uitgangspoorten*. Wordt een uitgangspoort hoog ("1"), dan wordt een spanning (+5 V) over de hierop aangesloten lijn naar het randapparaat gestuurd. Ook de uitgangspoorten zijn als registers te beschouwen en zijn samengevoegd tot een eenheid van acht bits.

De uit- en de ingangspoorten zijn niet in de normale geheugenruimte opgenomen. Omdat de proces-

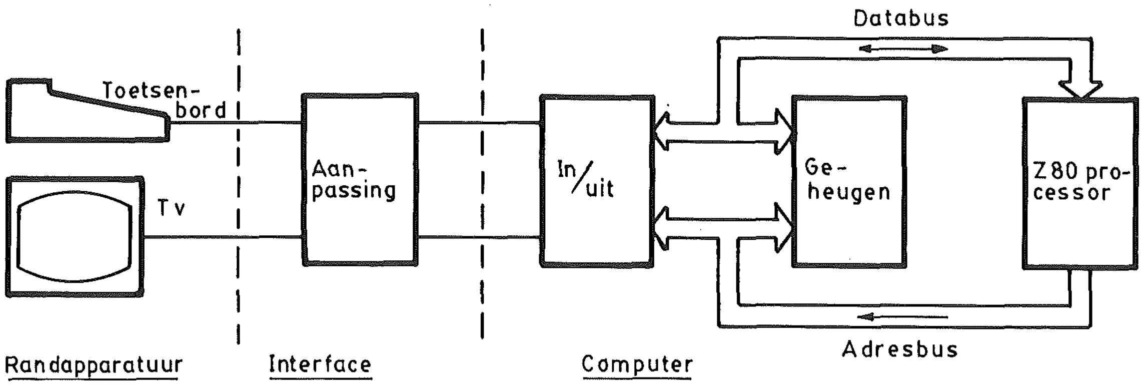


Fig. 1 Computersysteem.

sor echter steeds de gewenste uit- of ingangspoort moet kunnen vinden (adresseren) heeft elke poort wel zijn eigen adres gekregen. Er kunnen maximaal 256 (2^8) uitgangspoorten en ook maximaal 256 ingangspoorten worden toegepast. Voor het adresseren hiervan door de processor worden acht van de zestien adreslijnen gebruikt. Is een maximum aan ingangspoorten aanwezig dan worden hiervoor de adressen van 0 tot en met 255 gebruikt. Is ook een maximum aan uitgangspoorten aanwezig, dan worden hiervoor ook de adressen van 0 tot en met 255 gebruikt. Omdat er ook geheugenregisters met de adressen van 0 tot en met 255 zijn, zal het duidelijk zijn dat er nog iets moet gebeuren om aan te geven welk van de registers (uitgangspoort, ingangspoort of geheugenregister) nu wordt bedoeld als door de processor een adres in het gebied van 0 tot en met 255 wordt aangegeven. Hiervoor is nog een aantal besturingslijnen in uw computer aanwezig waarmee de processor een geheugenregister, of een uitgangspoort, of een ingangspoort kan activeren. Dat betekent dat de processor met het adres 150 zowel een geheugenregister als een uitgangspoort en een ingangspoort aanwijst, maar met de besturingslijnen toch slechts maar één van deze drie actief maakt.

Om het geheel nog eens samen te vatten is het blokschema in fig. 1 getekend. Hoewel het toetsenbord in dezelfde kast is gebouwd als uw computer wordt het toch als een randapparaat beschouwd. Ook de monitor (uw televisietoestel) is een randapparaat. Beide zijn verbonden met de in- en de uitgangspoorten van de computer. Nu kan dat niet rechtstreeks en daarom is er een aanpassing nodig, de *interface*. Ook deze aanpassing bevindt zich in de kast van uw computer. Verder ziet u in dit figuur dat de processor door middel van de adresbus zowel het geheugen als de in- en de uitgangspoorten kan adresseren. Van de adresbus die uit zestien geleiders bestaat gaan echter slechts acht geleiders naar de in- en de uitgangspoorten. Via de databus kan de processor gegevens sturen naar het geheugen en de uitgangspoorten maar ook gegevens lezen vanuit het geheugen en de ingangspoorten. De databus bestaat uit acht geleiders en kan daarom ook slechts binaire getallen van acht bits transporteren. De besturingslijnen zijn in dit schema niet aangegeven. Hierbij bevindt zich in elk geval een lijn die wordt gebruikt om het geheugen te activeren, een lijn voor het activeren van de in- en de uitgangspoorten, een lijn waarmee wordt aangegeven dat data naar een uitgangspoort of een geheugenelement moet worden geschreven en een lijn die aangeeft dat een ingangspoort of een geheugen moet worden uitgelezen.

Eerder is vermeld dat voor het "vertalen" van een BASIC programma in de voor de processor begrijpbare codetekens een programma nodig is, de BASIC interpreter. Dit programma is opgeslagen in geheugenelementen en neemt daarom *geheugenruimte* in beslag. Door middel van de ingangspoorten moet het toetsenbord, de cassetterecorder of de floppy disk worden uitgelezen terwijl ook gegevens naar de cassetterecorder, floppy disk of beeldscherm moeten worden gestuurd. Ook deze handelingen worden allemaal door de processor verricht, die daarvoor natuurlijk weer een programma nodig heeft. Dit programma wordt het *systeemprogramma* genoemd en neemt ook een gedeelte van de geheugenruimte in beslag. Direct na het in-

schakelen van uw computer wordt het systeemprogramma aan het werk gezet zodat u de randapparatuur meteen kunt gaan gebruiken. Zowel het systeemprogramma als de BASIC interpreter bevinden zich vast in uw computer en behoeven daar dus niet te worden ingevoerd. Dat betekent dat de inhoud van de geheugenelementen waarin deze programma's zijn opgeslagen niet verloren mag gaan bij het uitschakelen van de computer. Deze programma's zijn dan ook opgeslagen in een READ ONLY MEMORY (ROM). Dit is opgebouwd uit geheugenelementen waarvan de inhoud van de registers niet verloren kan gaan bij het uitschakelen van de computer en die niet kunnen worden ingeschreven, maar alleen kunnen worden uitgelezen. Verder is de geheugenruimte van de computer gevuld met geheugenelementen waarvan de registers wel kunnen worden ingeschreven en ook kunnen worden uitgelezen, het RANDOM ACCESS MEMORY (RAM). Deze geheugenelementen verliezen hun inhoud na het uitschakelen van de computer.

1.2. Het hexadecimale codesysteem.

De processor krijgt zijn instructies in de vorm van digitale getallen. Deze instructies worden uit het geheugen opgehaald via de databus en zijn daarom acht bits groot. Het moet mogelijk zijn om deze instructies via het toetsenbord in het geheugen te schrijven en het is lastig om dit binair te moeten doen. Dat is de reden waarom men daarbij een systeem hanteert waarbij het invoeren van een codegetal veel eenvoudiger is. Dit is het hexadecimale codesysteem, een zestientallig stelsel. Hierbij worden de binaire getallen "opgedeeld" in groepjes van steeds vier bits en elke groep wordt vervangen door een hexadecimaal cijfer. Met een groep van vier bits kunnen binaire getallen worden gevormd van 0000 tot en met 1111 (van 0 tot en met 15 decimaal). Dat betekent dat in het zestientallige stelsel steeds een cijfer te vinden is dat dezelfde waarde heeft als een bepaalde groep van vier bits. In het zestientallige stelsel hebben de "cijfers" namelijk een waarde die overeenkomt met 0 tot en met vijftien decimaal. Lijst 1 geeft de notatie van de cijfers uit het zestientallige of hexadecimale stelsel en de daarbij behorende binaire getallen.

Voor de zestien hexadecimale cijfers komen we aan de tien Arabische tekens van 0 tot en met 9 te kort. Deze worden daarom aangevuld met de letters A tot en met F.

Een getal van acht bits wordt een byte genoemd. Een byte wordt in twee groepen van vier bits verdeeld (tetrades, Engels: Nibbles). Een byte kan daarom worden uitgedrukt in hexadecimale cijfers. Het binaire getal 11010101 kan worden geschreven als D5H. Volgens lijst 1 komen de vier bits met de hoogste waarde (de hoge tetrade), 1101, overeen met DH en de vier bits met de laagste waarde (de lage tetrade), 0101, met 5H. Achter een hexadecimaal getal of een hexadecimaal cijfer is hier de letter H gebruikt om aan te geven dat het tot het hexadecimale stelsel behoort. Deze methode wordt veel toegepast bij de assemblerprogramma's die u bij het machinetaalprogrammeren nodig hebt. Zoals u weet wordt hiervoor bij BASIC &H voor het getal geplaatst.

Lijst 1. Hexadecimale getallen.

Hexa-decimaal	Binair	Decimaal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

Een geheugenadres is een binair getal van zestien bits (twee bytes) en kan daarom in vier tetrades worden gesplitst (vier groepjes van vier bits). Een adres wordt daarom steeds met vier hexadecimale cijfers genoteerd. Voorbeeld:
1001101000100001 komt overeen met 9A21H.
Adressen en geheugeninhouden zullen in dit boek steeds hexadecimaal worden gegeven. Waar geen verwarring mogelijk is zal het teken H achter het getal worden weggelaten.

De geheugenruimte is in *pagina's* verdeeld. Elke pagina omvat 256 adressen (bij de MSX computer wordt een blok van 16384 geheugenplaatsen, 16 Kbyte, wel een pagina genoemd, hetgeen in principe niet juist is). De totale geheugenruimte die de processor zonder meer kan adresseren omvat dan ook 256 pagina's (256×256=65536). Het paginanummer wordt weergegeven door de hoge byte van het adres. Zo omvat pagina 00H de adressen 0000H tot en met 00FFH (binair 0000000011111111), pagina 01H de adressen 0100H tot en met 01FFH, enzovoorts. Het hoogste paginanummer is FFH met de adressen FF00H tot en met FFFFH (binair 1111111111111111).

1.3. Indeling van het RAM geheugen.
Het RAM geheugen kan worden ingedeeld naar het gebruik dat er van wordt gemaakt:
a. Het datageheugen.
b. Het buffergeheugen.
c. Het stapelgeheugen of Stack.
d. Het programmageheugen.

a. Het datageheugen wordt gebruikt om gegevens in op te slaan die bij de verwerking van het programma moeten worden gebruikt. Deze gegevens zijn de ingevoerde variabelen, constanten en tabellen.

b. Het buffergeheugen wordt gebruikt om gegevens in te bewaren die sneller worden ingevoerd dan ze kunnen worden verwerkt. Om te voorkomen dat ze verloren gaan worden ze tijdelijk in een gedeelte van het geheugen opgeslagen dat dan als buffer dienst doet.

c. De Stack is het kladblaadje van de microcomputer. Hierin worden door de processor alle gegevens opgeslagen die tijdens het afwerken van het programma tijdelijk moeten worden bewaard.

d. In het programmageheugen wordt in codevorm het programma opgeslagen. De volgorde van de instructies in het geheugen is in het algemeen dezelfde als die in het programma. Het adres van de instructie die het eerst moet worden afgewerkt is het startadres van het programma; de volgende instructie heeft het naast hogere adres enz.

De volledige instructie van een machinetaalprogramma bestaat uit de bewerking of de handeling die de processor moet verrichten, de *operatie*, gevolgd door het getal of de variabele waarop die operatie betrekking heeft, de *operand*, dan wel het adres waarop die operand te vinden is, meestal een adres van het datageheugen. De operatie kan alleen in de vorm van een codegetal in het geheugen worden geplaatst. Dit getal is de *operatiecode*. Op de operatiecode volgt de operand dan wel het adres van die operand. In figuur 2 is het stroomdiagram weergegeven van een gedeelte van een programma. Een stroomdiagram is een blokschema waarin de vorm van een blok een bepaalde functie weergeeft. De blokken zijn door middel van een lijn met elkaar doorverbonden. Pijlen in de verbindingslijnen geven aan in welke richting het blokschema moet worden doorlopen. Uiteraard wordt het programma gestart bij het ovaal met het woord "start". In dit programmadeel volgen twee handelingen elkaar op. Indien twee of meer handelingen elkaar opvolgen wordt van een *sequentie* gesproken. In dit programmagedeelte worden de getallen 15H en 03H bij elkaar opgeteld. Het startadres is 9000.



Fig. 2 Stroomdiagram van een optelling.

Eerst moet het getal 15H worden opgehaald uit het datageheugen. Volgens de bij de Z80 geleverde tabel van de operatiecodes is de code voor de operatie "haal op" 3A. Bedenk dat adressen steeds hexadecimaal worden weergegeven. Dit zal ook het geval zijn met de operatiecoden! Nu moet het adres volgen van de operand, want die staat in het data-

geheugen. Dit adres is 9812 en bestaat dus uit twee bytes: de byte met de hoge plaatswaarde: 98 (MSB, Most Significant Byte) en de byte met de lage plaatswaarde: 12 (LSB, Least Significant Byte). Een geheugenplaats kan slechts één byte bevatten zodat voor het adres twee geheugenplaatsen nodig zijn. Voor de gehele instructie zijn drie bytes (en ook geheugenplaatsen) nodig, zodat dit een *drie bytes instructie* wordt genoemd. De gehele instructie luidt:

3A 12 98

Eerst de operatiecode met daarna het adres van de operand in de volgorde *lage* byte - *hoge* byte.
Dit is slechts de uitwerking van het eerste blok van figuur 2. Voor de optelling in het tweede blok geldt de operatiecode C6. In dit geval is de operand niet in het datageheugen geplaatst maar volgt direct op de operatiecode. De volledige instructie is daarom:

C6 03

en is dus een *twee bytes instructie*.
In figuur 3 is de situatie voor deze twee instructies in het geheugen geschetst. De geheugenplaatsen zullen steeds als vakjes worden voorgesteld waarin de inhoud van de geheugenplaats met een hexadecimaal getal is aangegeven. Het adres is voor het desbetreffende vakje geplaatst. De adressen zullen van beneden naar boven in de tekeningen, van laag naar hoog oplopen. De computer werkt dit programmadeel af door eerst op adres 9000 de operatiecode te lezen. Op de twee volgende adressen leest hij waar de operand te vinden is. Dan volgt de instructie voor het optellen van deze operand bij 03H.

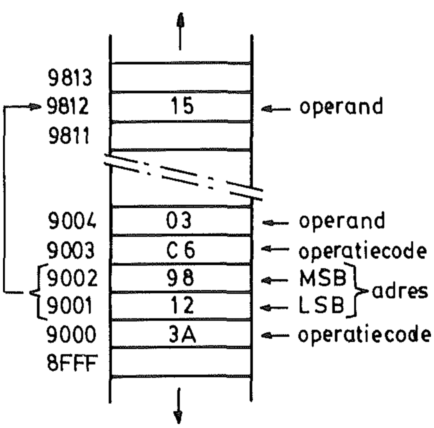


Fig. 3 Instructievolgorde in het geheugen.

1.4. Sprongopdrachten.

Het geheugen van een microcomputer is opgebouwd uit registers van acht bits en elk register is een geheugenplaats. Dat betekent dat de instructie in *code* in het geheugen moet worden gebracht (het geheugen wordt "geladen" met de instructies). Ook het laden van het geheugen loopt via de processor.
Veronderstel dat voor een bepaald programma vijf geheugenplaatsen gevuld moeten worden met 00H. De geheugenplaatsen zijn van 1 tot en met 5 genummerd (gp 1, gp2,...gp5). Het programma kan verlopen volgens fig. 4. De schrijfwijze voor: "laad gp 1 met 00H" is hier:

gp1 ← 00H

(de inhoud van gp1 wordt 00H).
Dit is weer een sequentie waarbij vijf dezelfde instructies worden gebruikt, alleen het adres (de geheugenplaats) is steeds anders. Het kan ook korter.

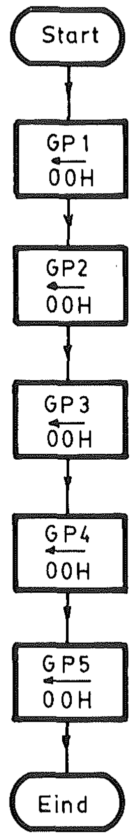


Fig. 4

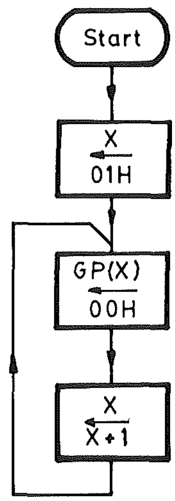


Fig. 5

Fig. 4 Sequentie.
Fig. 5 Onvoorwaardelijke sprong.

Bij het programma in fig. 5 wordt een bepaald register (bijvoorbeeld een geheugenplaats) aangeduid met X en gevuld met 01H. Dit register noemt men een *teller*. De volgende instructie laadt gp1 met 00H. Omdat X = 01H wordt gp(X) gelijk gp1. Daarna wordt de teller met 1 opgehoogd. De teller X heeft nu als inhoud dus 02H. Het programma gaat nu terug (maakt een *sprong*) naar de instructie die de aanduiding (*label*) "loop" heeft gekregen. Hiervoor moet een sprongopdracht in de vorm van een instructie (*Jump*) in het programma worden opgenomen, de computer doet niets uit zichzelf. Nu wordt geheugenplaats gp2 (X = 02H) met 00H geladen. Deze kringloop gaat zo door en ook de volgende geheugenplaatsen worden met 00H geladen. Nu is dit niet zo'n handig programma, want hoewel de eerste vijf geheugenplaatsen beslist hun inhoud krijgen, de computer gaat door totdat alle beschikbare geheugenplaatsen zijn behandeld. Na het vullen van de vijfde geheugenplaats moet de computer echter met dit programmadeel stoppen. Er is hier een *onvoorwaardelijke* of *ongeconditioneerde* sprong (Jump) toegepast terwijl een *voorwaardelijke* of *geconditioneerde* sprong (Branch) nodig is. Aan een voorwaardelijke sprong wordt, zoals de naam reeds zegt, een voorwaarde gesteld. Wordt aan de voorwaarde *niet* voldaan, dan wordt de sprong ook *niet* uitgevoerd. De voorwaarde die hier gesteld had moeten worden is: X < 6. Zodra de inhoud van het X-register de waarde 6 had gekregen was aan de voorwaarde niet meer voldaan en zou de sprong niet meer zijn uitgevoerd. Fig. 6 toont een iets andere manier voor het vullen van de geheugenplaatsen. Nu wordt de teller X met 05H geladen. De eerste geheugenplaats die nu geladen wordt is daarom gp5 (X = 05H). De teller wordt nu verminderd met 1. Het volgende blok (de ruit) stelt nu de voorwaarde: als X = 0 volgt *geen* sprong. Zover is het nog niet (X = 04H). Het programma gaat dus terug naar "loop" en gp4 wordt geladen. Dit gaat zo door tot X = 01. Na het laden van gp1 als laatste geheugenplaats wordt de teller opnieuw met 1 verminderd en is dus 0. Er wordt daardoor niet meer naar "loop" teruggegaan en het programma is beëindigd. Het "ja" en het "nee" bij de uitgangen van het blok dat de voorwaarde stelt geven aan in welke richting (terug of verder) het programma bij een bepaalde situatie wordt voortgezet.

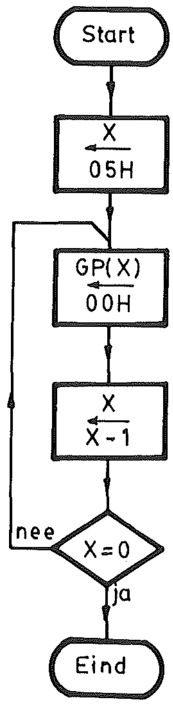


Fig. 6 Voorwaardelijke sprong.

2. Rekenkundige en logische functies

2.1. Binair optellen.

Het woord "computer" is afgeleid van "to compute" (rekenen) en het zal dan ook niemand verbazen dat er met dat ding kan worden gerekend. We hebben dan ook gezien dat in BASIC alle rekenkundige berekeningen (en nog andere functies) kunnen worden uitgevoerd met een zeer grote nauwkeurigheid. Toch kan de processor zelf alleen maar optellen. Als u echter bedenkt dat een vermenigvuldiging kan worden uitgevoerd door herhaald optellen $2 \times 5 = 5 + 5$) dan ziet u dat andere bewerkingen dan optellen door middel van een passend programma kunnen worden uitgevoerd. Misschien lijkt dat wat omslachtig, maar de processor is zo snel dat dat geen bezwaar kan zijn.

De processor in uw computer kan slechts binaire getallen van acht bits optellen, per keer niet meer dan twee getallen. De optelling vindt plaats tussen twee registers. Hoe ook grotere getallen kunnen worden opgeteld wordt bij de verklaring van de rekeninstructies behandeld. De nu volgende optelling van twee achtbits getallen geeft geen moeilijkheden:

```
01101100
10001011 +
----------
11110111
```

Anders is dat met de volgende optelling:

```
0   11011001
0   10110110 +
-   ----------
1   10001111
```

U ziet dat het resultaat van de optelling groter is dan acht bits en niet meer in een normaal register past. Om een dergelijke optelling te kunnen uitvoeren wordt het hoogste bit van het resultaat, het zogenaamde *"Carrybit"*, in een speciaal register bewaard. In de processor bevinden zich, net als in het geheugen, ook een aantal registers. Speciaal voor het bewaren van een aantal bijzondere bits is hierin een achtbitsregister aanwezig, het *processorconditieregister* of ook wel *FLAG register* genoemd, waarin onder andere het Carrybit een speciale plaats heeft. Bij de optelling in het eerste voorbeeld is er geen Carrybit. De plaats in het FLAG register hiervoor wordt dan 0. Bij de optelling in het tweede voorbeeld is er een Carrybit. De desbetreffende plaats in het FLAG register wordt nu 1.

2.2. Negatieve getallen.

In de "gewone" rekenkunde zijn we gewend om negatieve getallen met een minteken aan te geven. Bij gebruikmaking van registers, zoals in computers, werkt dat anders.
Op een cassettedeck (of een datarecorder) bevindt zich in het algemeen een drie-cijferige bandteller. Als u deze op nul stelt en daarna de opwindspoel één omwenteling terug draait, dan komt de teller op 999 te staan. Dat is kennelijk de stand die met -1 overeenkomt. Iets dergelijks kennen we bij de registers in de computer ook. Als we van een getal in een register dat de waarde 00000000 heeft 1 aftrekken dan is het resultaat:

```
00000000
00000001 -
----------
11111111
```

Als we deze berekening op de juiste manier hadden uitgevoerd, dan was hij als volgt verlopen:

```
C   76543210 bitnummer

1   00000000
0   00000001 -
-   ----------
0   11111111
```

In dit geval vindt het omgekeerde plaats van bij het optellen: als we in een register een bit "te kort" komen, dan wordt dat geleend uit de plaats van het Carrybit uit het FLAG register. Er wordt dan gesproken van een *"Borrow"*. De plaats in het FLAG register voor het Carrybit of de Borrow zullen we in het vervolg steeds met "C" aangeven.

Uit het voorgaande blijkt dat het binaire getal 11111111 dezelfde waarde kan hebben als het getal

– 1. Er zijn twee manieren om een register te gebruiken:

a. Voor uitsluitend positieve getallen (00000000 - 11111111)
b. Voor zowel positieve als negatieve getallen.

Wordt een register voor zowel positieve als negatieve getallen gebruikt, dan is een methode nodig om deze van elkaar te kunnen onderscheiden. Hiervoor wordt het hoogstwaardige bit (bit 7) van het register gebruikt. Is dit bit een 0, dan is het getal in het register positief. Het grootste positieve getal dat dan mogelijk is heeft de waarde 01111111. Dit komt overeen met 127 decimaal. Is het hoogstwaardige bit 1 dan is het getal in het register een negatief getal. De in absolute zin grootste waarde hiervan is 128. Dit volgt uit de volgende berekening:

C	76543210 bitnummer
1	00000000
0	10000000 –
0	10000000

Voegen we de Borrow (C) bij het aftrektal, dan wordt dit 100000000 (256 decimaal). Trekken we hiervan 10000000 (128 decimaal) af, dan is het resultaat 10000000, hetgeen dus -128 decimaal moet voorstellen. Elke grotere aftrekker maakt het bit 7 van het resultaat 0.

Samenvattend kan worden gesteld dat een register positieve getallen van 0 tot en met 127 binair kan bevatten en negatieve getallen van – 1 tot en met – 128 binair.

Het is betrekkelijk eenvoudig om van een positief binair getal de negatieve waarde te vinden. Stel het getal is 01011101. Hiervan worden alle bits geïnverteerd: 10100010. Nu wordt er nog een 1 bij opgeteld: 10100011. Dit resultaat is de negatieve waarde van 01011101. U zult zien dat de volgende berekening hetzelfde resultaat geeft:

C	76543210 bitnummer
1	00000000
0	01011101 –
0	10100011

Hoewel de processor een aantal instructies kent voor het aftrekken van twee getallen voert hij deze aftrekkingen steeds uit door van de aftrekker de negatieve waarde te bepalen en dan deze bij het aftrektal op te tellen. Stel dat de volgende berekening door de processor moet worden uitgevoerd:

C	76543210 Bitnummer
1	01110100 Aftrektal
0	01001101 Aftrekker
1	00100111 Verschil

De volgende handelingen worden door de processor verricht: C = 1; aftrekker 01001101. Inverse van de aftrekker + 1: C = 0; 10110011. Hierna volgt de optelling:

C	7654321 bitnummer
0	01110100
0	10110011 +
1	00100111

Deze rekenmethode wordt het *"twee-complement-systeem"* genoemd.

Voor bepaalde instructies is het voor de processor nodig om het teken (positief of negatief) van het resultaat van een voorafgaande bewerking te kennen. Daarom wordt dit teken bewaard als een bit van het FLAG register. Het desbetreffende bit wordt "S" genoemd (van SIGN) en is niet anders dan de waarde van bit 7 van het resultaat van de bewerking. Na de voorgaande berekening is S dus 0. Het kan met dit tekenbit ook wel eens fout gaan bij bepaalde berekeningen. Bij de volgende berekening worden twee positieve getallen bij elkaar opgeteld:

Getal a: 01110010; getal b: 00110110

C	76543210 Bitnummer
0	01110010 Getal a
0	00110110 Getal b
0	10101000 C = 0; S = 1

Dit is een optelling van twee positieve getallen, maar als we S moeten geloven, dan is het resultaat negatief. Dat kan natuurlijk niet. Om aan te geven

dat de inhoud van S niet juist is en er dus bij de berekening een overdracht is geweest (Overflow) van bit 6 naar bit 7, zoals dat ook hier het geval is, is er in het FLAG register nog een FLAG aanwezig, het Overflow bit (V). Is er een overdracht van bit 6 naar bit 7 geweest, dan is de Overflow FLAG 1, waarmee wordt aangegeven dat de aanduiding van het teken in S niet juist is.

2.3. De BCD-code.
De registers in een microcomputer zijn steeds zodanig van lengte (vierbits, achtbits, zestienbits enzovoorts) dat zij in groepjes van vier bits zijn in te delen. Dit maakt het toepassen van het hexadecimale codesysteem mogelijk. Zoals u weet is dat een zestientallig stelsel. Willen we echter werken in het tientallige stelsel, dus in principe zonder om te rekenen naar en van het tweetallige stelsel, dan is het toepassen van de BCD-code nodig. In de BCD-code (Binery-coded decimal) wordt aan elk cijfer van een getal de overeenkomstige binaire waarde toegekend. Zo zal het getal 7251 moeten worden omgezet tot 0111 0010 0101 0001, hetgeen in een zestienbits register genoteerd wordt als 0111001001010001. Uiteraard is de binaire waarde van dat getal niet in overeenstemming met 7251 (binaire waarde: 29265). Op overeenkomstige wijze kan een in de BCD-code geschreven getal worden teruggevonden. Vinden we in een register het getal 1000010010010011, dan verdelen we dat eerst in groepjes van vier bits (tetrades of Nibbles): 1000 0100 1001 0011 en vinden daaruit dat het de voorstelling is van het decimale getal 8493.

2.4. Het rekenen in de BCD-code.
Omdat de BCD-code nu eenmaal geen getallenstelsel is zoals het binaire getallenstelsel, gaat in het algemeen het rekenen in de BCD-code op een andere wijze dan het binair rekenen, tenminste als het resultaat groter is dan 10 decimaal. De methode die gebruikt moet worden wordt op voldoende wijze gedemonstreerd als we ons beperken tot bewerkingen met getallen van maximum acht bits. De volgende twee voorbeelden laten zien dat het optellen geen moeilijkheden geeft als het resultaat niet groter is dan 9 decimaal.

decimaal	BCD
03	0000 0011
04 +	0000 0100 +
07	0000 0111

decimaal	BCD
06	0000 0110
03 +	0000 0011 +
09	0000 1001

Dit is anders als de som groter is dan 9 decimaal:

decimaal	BCD
06	0000 0110
05 +	0000 0101 +
11	0000 1011

Dit laatste voorbeeld is niet juist omdat volgens de optelling 6 + 5 = 11 het resultaat in BCD-code 00010001 had moeten zijn. Het juiste getal wordt gevonden als bij de waarde 00001011 nog eens 00000110 wordt opgeteld:

decimaal	BCD
06	0000 0110
05 +	0000 0101 +
11	0000 1011
correctie	0000 0110 +
	0001 0001

Een tweede voorbeeld:

decimaal	BCD
09	0000 1001
04 +	0000 0100 +
13	0000 1101
correctie	0000 0110 +
	0001 0011

Er zijn nog andere gevallen waarbij ook een correctie nodig is, bijvoorbeeld als het resultaat groter is dan 15:

decimaal	BCD
09	0000 1001
08 +	0000 1000 +
17	0001 0001

Het zal duidelijk zijn dat ook nu een correctie nodig is. Aan de waarde van de laagste tetrade kan de computer dat echter niet waarnemen. Nu heeft

echter een overdracht plaatsgevonden van de eerste tetrade naar de tweede tetrade (van bit 3 naar bit 4). Dit is een reden om ook een correctie toe te passen:

```
0001  0001
0000  0110+
----------
0001  0111
```

Of al dan niet een overdracht van de eerste naar de tweede tetrade heeft plaatsgevonden wordt in de computer vastgelegd in een bit van het FLAG-register, de Half Carry (H). Alleen als de overdracht heeft plaatsgevonden krijgt H de waarde 1.

Wat geldt voor de eerste tetrade geldt ook voor de tweede tetrade. Ook nu moet worden gecorrigeerd als het resultaat in deze tetrade groter is dan 9 of als er een overdracht is naar de naast hogere tetrade. Deze laatste overdracht wordt, evenals bij binair rekenen, genoteerd als C=1.

Voorbeelden:

```
decimaal   BCD
023        0010  0011
095+       1001  0101+
----       ----------
118        1011  1000 C=0 H=0
correctie  0110  0000+
           ----------
           0001  1000 C=1 H=0

decimaal   BCD
058        0101  1000
089+       1000  1001+
----       ----------
147        1110  0001 C=0 H=1
correctie  0110  0110+
           ----------
           0100  0111 C=1

decimaal   BCD
098        1001  1000
089+       1000  1001+
----       ----------
187        0010  0001 C=1 H=1
correctie  0110  0110
           ----------
           1000  0111 C=1
```

Bij het laatste voorbeeld zal op de aanwijzingen C=1 en H=1 de processor beide tetraden corrigeren. De optelling van de correctie zou normaal te betekenen hebben dat geen Carry zou plaatsvinden. In dit geval wordt echter de Carry van de eerste bewerking overgenomen, C=1.

Stel dat de hoge tetrade van het resultaat van een optelling n is en de lage tetrade m, dan is dit resultaat n,m. Voor het binaire getal:

```
7 6 5 4   3 2 1 0 bitnummer
1 1 0 1   0 1 1 0
```

is n dus 1101 en m 0110. Lijst 2 geeft voor de diverse mogelijkheden van n, m, C en H voor de correctie, het resultaat na de correctie.

Lijst 2. Correcties voor BCD-optellingen.

C	h.tetrade	H	l.tetrade	Resultaat
0	n<10	0	m<10	n,m C=0
0	n<10	1	m<10	n,m+6 C=0
0	n<9	0	m>9	n+1,m+6 C=0
1	n<10	0	m<10	n+6,m C=1
1	n<10	1	m<10	n+6,m+6 C=1
1	n<10	0	m>9	n+6+1,m+6 C=1
0	n>9	0	m<10	n+6,m C=1
0	n>9	1	m<10	n+6,m+6 C=1
0	n>8	0	m>9	n+6+1,m+6 C=1

Het aftrekken met BCD-getallen vindt plaats in het 10-complementsysteem. Dit betekent: de aanvulling tot het getal 10 of tot een macht van 10. Zo is het 10-complement van 3 gelijk aan 7 en het 10-complement van 63 gelijk aan 37. Voor een aftrekking wordt eerst het 10-complement van de aftrekker bepaald en dit opgeteld bij het aftrektal. Stel: we willen de aftrekking 63 − 19 = 44 uitvoeren in het 10-complementsysteem. Het 10-complement van 19 is 81 (100 − 19):

```
decimaal   BCD
63         0110  0011
81+        1000  0001+
----       ----------
44         1110  0100 C=0 H=0
correctie  0110  0000
           ----------
           0100  0100
```

Alle handelingen voor een aftrekking met twee BCD-getallen worden door de Z80 processor zelf uitgevoerd en behoeven dus niet te worden geprogrammeerd. Hij voert de bewerking geheel op zijn

eigen wijze uit en gebruikt daarbij een extra bit in het FLAG- register, het bit N.

2.5. Logische bewerkingen.

Logische bewerkingen zijn nu juist die bewerkingen die bij uitstek geschikt zijn om door de processor te worden uitgevoerd. De bewerkingen die hij kan uitvoeren zijn AND, OR en XOR. De logische bewerkingen worden uitgevoerd tussen de overeenkomstige bits uit twee registers (bit 0 van register a met bit 0 van register b, bit 1 van register a met bit 1 van register b enz.). Wellicht ten overvloede volgen hier de definities van de bewerkingen:
Het resultaat van een AND-bewerking tussen twee bits is slechts dan 1 als beide bits 1 zijn.
Het resultaat van een OR-bewerking tussen twee bits is slechts dan 0 als beide bits 0 zijn.
Het resultaat van een XOR-bewerking tussen twee bits is slechts dan 1 als beide bits ongelijk zijn.

Voorbeeld van een AND-bewerking:

10101010
11001100 AND
———————
10001000 resultaat.

Voorbeeld van een OR-bewerking:

10101010
11001100 OR
———————
11101110 resultaat.

Voorbeeld van een XOR-bewerking:

10101010
11001100 XOR
———————
01100110 resultaat.

De tekens die voor de logische bewerkingen worden gebruikt zijn:
AND: $\wedge$
OR: $\vee$
XOR: $\veebar$

2.6. Basisprogramma's voor logische functies.

De vorige paragraaf handelde over logische bewerkingen tussen twee bits. Vanuit BASIC bent u mogelijk meer gewend aan logische bewerkingen tussen uitspraken: IF A>B AND B=C THEN... Hierin is A>B een uitspraak die waar (1) of onwaar (0) kan zijn. Als we beweren (uitspreken) dat A>B terwijl A=5 en B=6, dan is deze uitspraak uiteraard niet waar. Ook B=C is een uitspraak. Het betreft hier uitspraken over variabelen (A, B en C). We kunnen ook een uitspraak doen over de waarde van een variabele, zoals A=6 of A=0. Als we stellen A=6 terwijl A in werkelijkheid 5 is, dan is de uitspraak onwaar. In fig. 6 hebben we gezien dat een voorwaardelijke sprong gemaakt kan worden afhankelijk van de vraag of een bepaalde uitspraak (hier X=0) waar of onwaar is. In deze figuur wordt een sprong gemaakt als de uitspraak X=0 niet waar is. Dat betekent dat we de werkelijke waarde van X (de inhoud van een register X) moeten toetsen aan de uitspraak X=0.

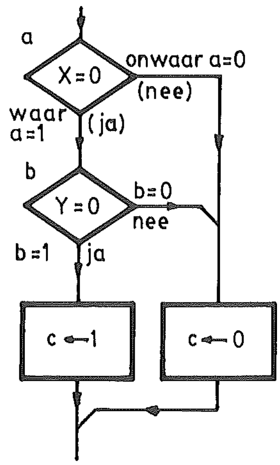


Fig. 7. Basisprogramma voor een AND-functie.

In fig. 7 is een programmadeel getekend waarin drie registers voorkomen. Van dit programmadeel is een waarheidstabel op te stellen waarin een 1 genoteerd wordt voor een *ware* bewering en waarin een 0 genoteerd wordt voor een *onware* bewering. De 1 en de 0 in deze tabel hebben dus geen betrekking op de werkelijke inhoud van de bedoelde registers (register X en register Y) maar hebben betrekking op het al of niet waar zijn van de *uitspraak* over de inhoud van de registers. Onder a (eerste blok van het programmadeel) is de uitspraak: X=0. Stel de inhoud van register X is werkelijk 0. De *uitspraak* X=0 is dan waar en er wordt een 1 in de waarheidstabel genoteerd. Ondanks het feit dat de *inhoud* van het register 0 is wordt dus toch een 1 in de tabel gevonden. In fig. 7 betreft het twee uitspraken: X=0 (a) en Y=0 (b). In de waarheidstabel wordt in kolom c voor elk

van de vier mogelijke combinaties van a en b aangegeven wat de inhoud van register c zal zijn. Deze kolom geeft dus de *werkelijke* inhoud van register c weer. Er is dan ook sprake van een resultaat en niet van een uitspraak.

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

Dit is een tabel voor de functie $c = a \wedge b$.

Ook voor het programmadeel in fig. 8 is een waarheidstabel op te stellen:

a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

Deze tabel is voor de functie $c = a \vee b$, zodat met het programmadeel in fig. 8 een OF-functie is te verwezenlijken.

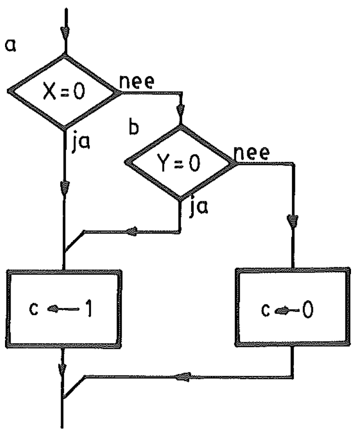


Fig. 8. Basisprogramma voor een OF-functie.

Uiteraard is het ook mogelijk de functie $c = a \veebar b$ te realiseren. Hiertoe dient het programmadeel in fig. 9. In dit figuur zijn twee punten gemerkt door een cirkel met een cijfer, in dit geval een 1. Tussen twee gelijk genummerde punten is een verbinding aanwezig. Deze methode wordt gevolgd om onduidelijkheden door elkaar kruisende lijnen en dergelijke te voorkomen.

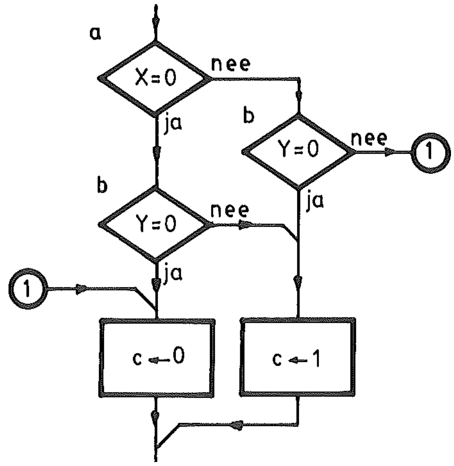


Fig. 9. Basisprogramma voor de XOF-functie.

De waarheidstabel voor fig. 9 is:

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Ook niet-functies zijn te realiseren. Het programmadeel in fig. 10 geeft de realisatie van de functie NOT($a \wedge b$):

a	b	$a \wedge b$	c
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

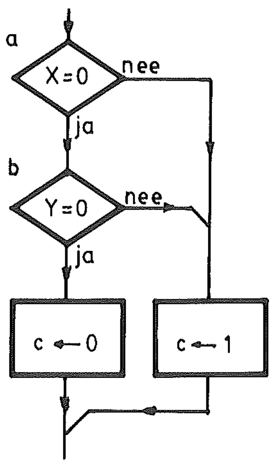


Fig. 10. Basisprogramma voor een NIET-functie.

Merk op dat het enige verschil met fig. 7 is dat de inhoud van register c is verwisseld.

Fig. 11 geeft de oplossing voor een samengestelde functie:

$d = (a \wedge b) \vee c$.

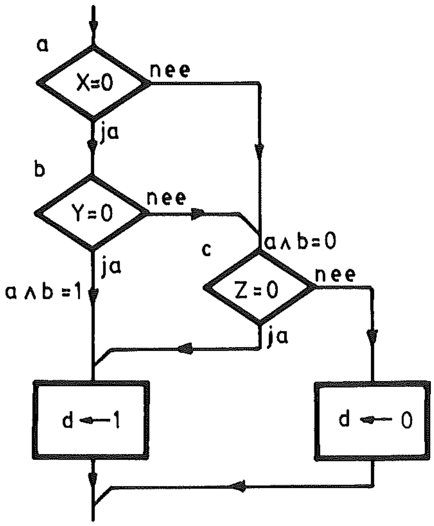


Fig. 11. Basisprogramma voor een samengestelde functie.

De waarheidstabel voor dit programmadeel kent drie variabelen. Er zijn dus $2^3 = 8$ mogelijkheden. Eerst wordt de tabel voor $a \wedge b$ gemaakt, waarna een OF-functie wordt gerealiseerd tussen deze en de tabel voor c.

a	b	c	$a \wedge b$	d
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

Bij deze en ook bij vorige voorbeelden zijn steeds de uitspraken X = 0, Y = 0 enzovoorts genomen. Voor een voorwaardelijke sprong kan echter elke uitspraak worden genomen, zoals
X = positief;
X = negatief;
X = n (n is een getal);
Carry = 0;
Carry = 1; enzovoorts.

3. De Z80-processor

3.1. Inleiding.

Tot nu toe hebben we met een aantal bewerkingen kennis gemaakt die de processor moet kunnen uitvoeren en met een aantal basisprogramma's, zoals de sequentie, onvoorwaardelijke sprong en de voorwaardelijke sprong. Om deze bewerkingen te kunnen verrichten moeten de volgende handelingen mogelijk zijn:

a. De processor moet het juiste adres op de adresbus kunnen plaatsen.
b. De processor moet data van de databus kunnen innemen.
c. De processor moet een operatiecode kunnen decoderen en de diverse besturingsopdrachten kunnen geven.
d. De processor moet rekenkundige en logische bewerkingen kunnen uitvoeren.

Dat de processor in elk geval de reeds genoemde bewerkingen moet kunnen uitvoeren wil nog niet zeggen dat hij niet tot nog meer in staat is. Welke bewerkingen allemaal mogelijk zijn zal in dit en in de volgende hoofdstukken duidelijk worden.

3.2. Het adresseren.

In fig. 1 hebben we gezien dat voor het transport van de data in de computer een databus nodig is die is opgebouwd uit acht geleiders, voor elk bit van een ''woord'' (binair getal) één. Om de geheugenplaatsen en ook de in- en de uitgangspoorten te kunnen adresseren is een zestienlijns adresbus nodig. Ook binnen de processor is datatransport tussen de diverse onderdelen nodig en moeten diverse stuurcommando's kunnen worden gegeven. De geleiders hiervoor vatten we samen onder de naam *''interne bus''*. Deze interne bus is van de adresbus gescheiden door middel van een speciaal register, bestaande uit de *adresbuffers*. Deze adresbuffers bevinden zich in de processor. De uitdrukking ''buffer'' wordt wel gebruikt voor een register waarin tijdelijk data kan worden opgeslagen, maar ook voor schakelingen die in staat zijn aan de belasting (in dit geval de adresbus) voldoende energie te leveren. De adresbuffers vormen de scheiding tussen de interne bus en de adresbus en er zijn er zestien van nodig. Het adres dat op de adresbus wordt geplaatst wordt steeds via de interne bus in de adresbuffers geladen (fig. 12).

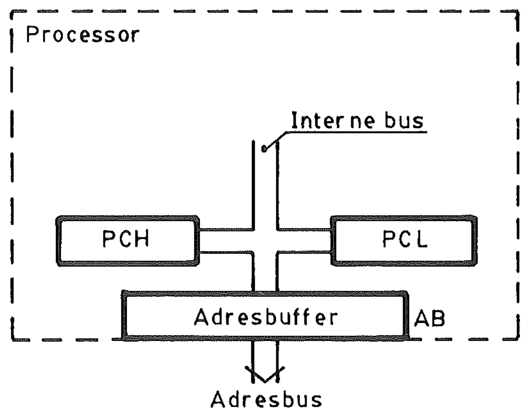


Fig. 12. De adresbuffers.

Het adres kan van verschillende registers in de processor afkomstig zijn. Een ''speciaal'' adresregister is de *programmateller* (Program Counter, PC). Dit is een zestienbits register dat in fig. 12 in twee achtbitsregisters gescheiden is getekend, één voor de lage byte (PCL) en één voor de hoge byte (PCH). De programmateller vormt een belangrijke functie bij het doorlopen van een sequentie. Bij aanvang van een programma bevat de programmateller het eerste adres van dit programma, het *startadres*. Op dit adres staat de operatiecode van de eerste instructie van dit programma. Nadat het startadres in de adresbus is geladen wordt de programmateller met 1 opgehoogd: $PC \leftarrow PC+1$. Zonodig kan dit adres voor de volgende handeling (de tweede instructie van het programma) op de adresbus worden geplaatst door het in de adresbuffers te laden. Dit geheel herhaalt zich voor elke programma-instructie. Op deze manier wordt een programma, of een gedeelte van een programma, geheugenplaats na geheugenplaats geadresseerd. Voor het maken van een voorwaardelijke of een onvoorwaardelijke sprong wordt de programma-

teller met het nieuwe adres geladen van waaruit dan de volgende sequentie kan worden doorlopen. Nu zijn het niet alleen programma-adressen die op de adresbus moeten worden geplaatst. De adresbuffers moeten ook geladen kunnen worden met een adres van een variabele. Daarom kunnen de adresbuffers ook geladen worden met de inhoud van een ander register uit de processor dan de programmateller.
Voor het doorlopen van het programma van fig. 2 moeten de adresbuffers na elkaar de volgende (hexadecimale) adressen bevatten:

9000 - adres 1e opcode
9001 - LSB
9002 - MSB
9812 - adres 1e operand
9003 - adres 2e opcode
9004 - adres 2e operand

In deze reeks is 9812 het adres dat niet uit de programmateller afkomstig is maar door de processor uit de geheugenplaatsen 9001 en 9002 is gelezen. Daarna wordt het in de adresbuffers geladen. De andere adressen (9000 tot en met 9004) zijn wel afkomstig van de programmateller.

3.3. De databuffers.

Nadat een adres via de adresbuffers op de adresbus is geplaatst, kan de data die zich op dat adres bevindt worden gelezen. Dit kan data zijn die zich in een geheugenregister bevindt maar ook data die zich in een ingangspoortregister bevindt. Het kan echter ook zijn dat data vanuit een processorregister moet worden geschreven in een geheugenregister of in een uitgangspoortregister. Deze handelingen worden verricht via de databuffers in de processor. Deze vormen de scheiding tussen de interne bus en de databus. De databuffers moeten anders van constructie zijn dan de adresbuffers. Zij moeten in staat zijn om in twee richtingen te werken, data moet op de databus geplaatst kunnen worden (schrijven) maar moet ook van de databus in de processor kunnen worden gevoerd (lezen).
Niet altijd wordt data die vanuit een geheugenregister of een ingangspoortregister op de databus wordt geplaatst, ook door de databuffers aan de interne bus doorgegeven. Of dat gebeurt hangt af van een stuursignaal uit de controle- en besturingslogica in de processor.

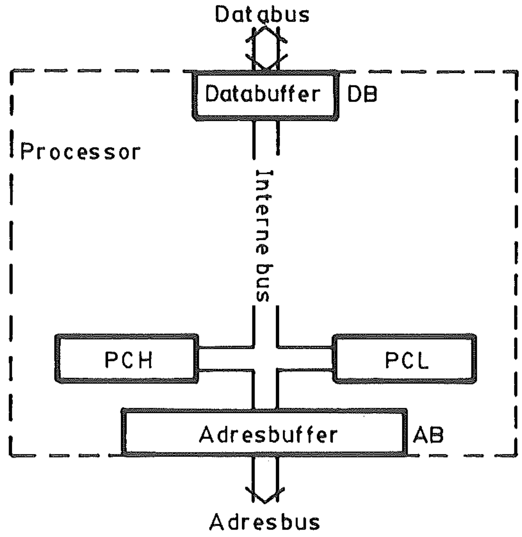


Fig. 13. De databuffers.

3.4. Besturing en decodering.

Reeds is vermeld dat een programma bestaat uit een opeenvolging van instructies. De eerste code van een instructie is steeds de operatiecode. Als het adres van de instructie vanuit de programmateller in de adresbuffers is geladen dan vinden we de operatiecode op de databus. Deze data wordt vanuit de databuffers en via de interne bus in het *instructieregister* geladen. Door de *besturings- en decoderingslogica* wordt de operatiecode gedecodeerd. Daarbij wordt vastgesteld welke handelingen door de processor moeten worden verricht om de operatie die door de code wordt bedoeld, te kunnen uitvoeren. Hierbij speelt de *interne besturing* een belangrijke rol. Deze regelt de opeenvolging van de diverse werkingen en van de datastroom binnen de processor. Hierdoor wordt bijvoorbeeld bepaald of een nieuw adres in de adresbuffers moet worden geladen en vanuit welk processorregister dat dient te geschieden (programmateller of ander register). Ook regelt de interne besturing de verbinding tussen een bepaald processorregister en de databuffers en maakt de databuffers tot in- of uitgangsregisters. Verder zorgt de interne logica ervoor dat de juiste rekenkundige of logische bewerkingen door de processor worden uitgevoerd.
Indien buiten de processor iets dient te gebeuren (selecteren van geheugen, in- of uitgangspoorten; lezen uit een register of schrijven naar een register), dan zorgt hiervoor de externe besturing die ook alweer vanuit de processor wordt geregeld.

De besturingslogica is erop ingesteld dat na het afwerken van een instructie door de processor steeds het adres van de volgende instructie op de databus verschijnt en dat de data die daarbij hoort wordt doorgegeven aan het instructieregister, waarmee de voortgang van het programma wordt gegarandeerd.

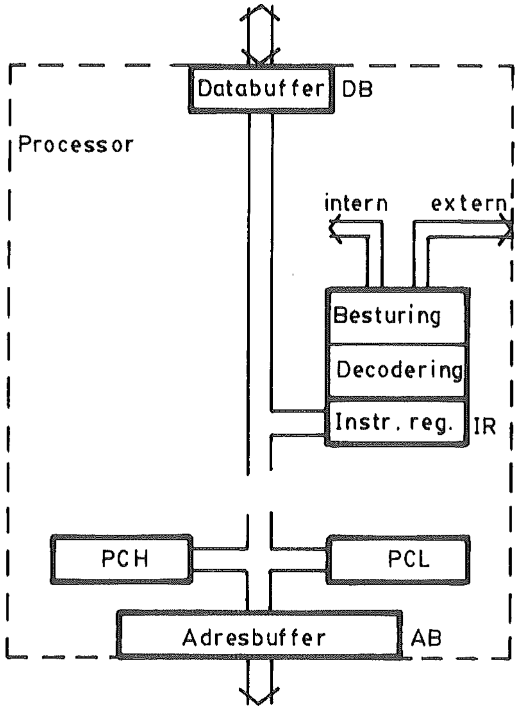


Fig. 14. Besturing en decodering.

3.5. De rekenkundige en logische eenheid.
Als de operatiecode van een rekenkundige of een logische bewerking zich in het instructieregister bevindt, dan wordt onder invloed van de besturingslogica de *Arithmetic and Logic Unit* (ALU) ingeschakeld (fig. 15). Gaan we uit van het voorbeeld bij fig. 3, dan moeten de getallen 15H en 03H bij elkaar worden opgeteld. Voordat de operatiecode voor ''optellen'' (C6) in het instructieregister wordt gebracht moet het eerste getal van de bewerking (15H) al in een speciaal register zijn geladen, de *Accumulator*, hier verder aangeduid met *Accu* (A). De Accu is een van de meest belangrijke registers in de processor. Het wordt in elk geval gebruikt bij de meeste bewerkingen die door de processor moeten worden uitgevoerd (fig. 16). De ope-

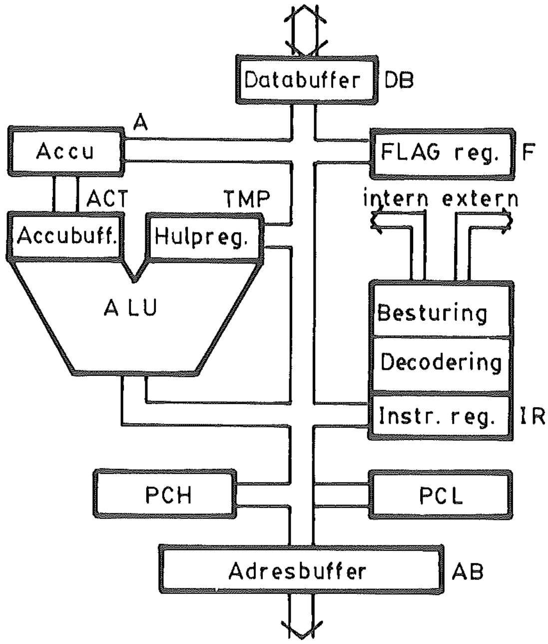


Fig. 15. De rekenkundige en logische eenheid.

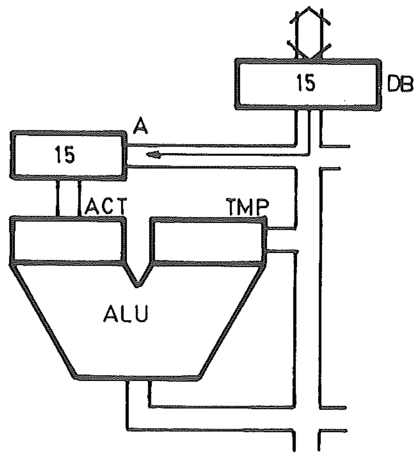


Fig. 16. Optelling, eerste operand in A.

ratiecode voor optellen heeft de volgende handelingen tot gevolg:

a. Het getal van de tweede operand (03H) wordt in een *hulpregister* (TMP) geladen en het getal van de eerste operand (15H) wordt vanuit de Accu in de *Accubuffer* ACT geplaatst (fig. 17).

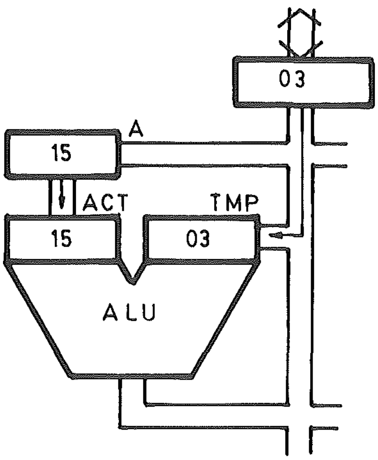


Fig. 17. Optelling, tweede operand in TMP.

b. De getallen in ACT en TMP worden bij elkaar opgeteld. Het resultaat (18H) wordt via de interne bus in de ACCU geplaatst. Dat betekent dat de oorspronkelijke inhoud van de ACCU (15H) verloren gaat (fig. 18).

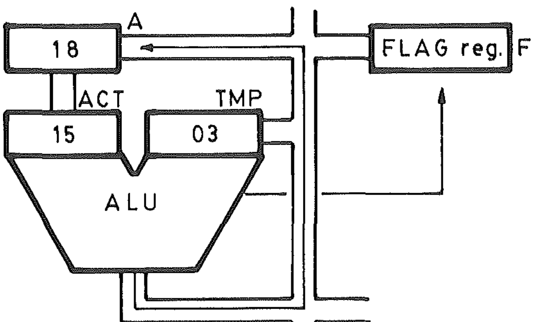


Fig. 18. Optelling, het resultaat in A.

Afhankelijk van het resultaat zullen een aantal bits van het FLAG-register F worden geset (''1'' worden) of worden gereset (''0'' worden). Dit zijn in elk geval de bits C (Carry), S (Sign), H (Half Carry) en V (Overflow) en bij het rekenen in de BCD-code het bit N, geheel in overeenstemming met hetgeen in hoofdstuk 2 is behandeld.

3.6. De werk- en hulpregisters.

In fig. 19 zijn de voornaamste onderdelen van de Z80-processor gegroepeerd rond de interne bus. Deze paragraaf beschrijft de registers die nog niet zijn behandeld.

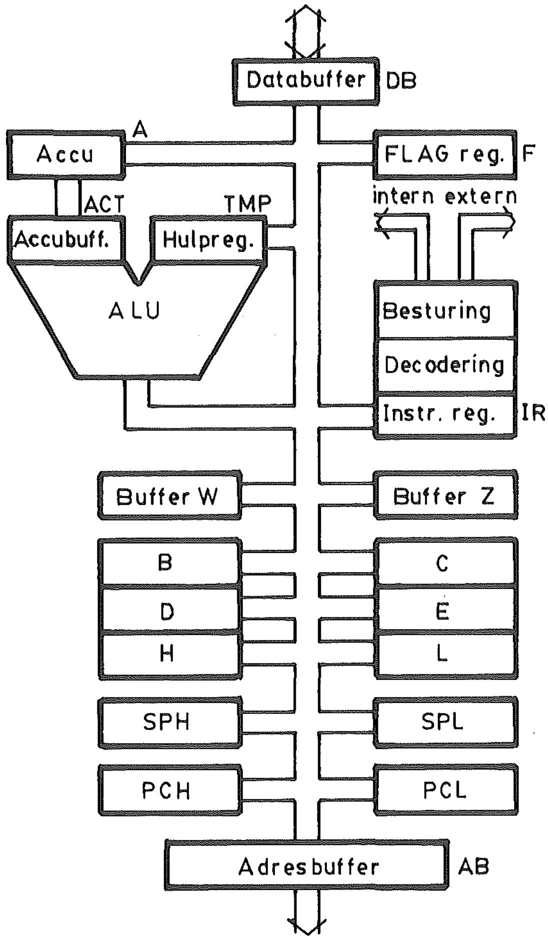


Fig. 19. De Z80, interne organisatie.

Het FLAG-register (F) zijn we al eens eerder tegengekomen. Men noemt het ook wel het processorconditieregister, conditiecoderegister of het statusregister. De bedoeling van dit register is dat een bepaald bit wordt geset of gereset afhankelijk van het resultaat van een bepaalde bewerking door de ALU. De volgende bits van dit register worden gebruikt (fig. 20):

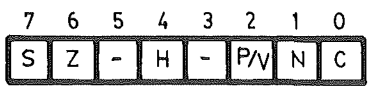


Fig. 20. Het FLAG register.

Bit 0. Dit bit wordt met ''C'' aangeduid en wordt onder andere gebruikt voor het Carrybit bij optellingen en Borrow bij aftrekkingen. Er zijn nog meer instructies die de waarde van dit bit kunnen

bepalen. Bij de behandeling van de desbetreffende instructie zal hierop worden gewezen.
Bit 1. Dit bit wordt aangeduid met "N" en wordt door de processor gebruikt bij het rekenen in de BCD-code. Bij het optellen in de BCD-code wordt dit bit 1 en bij het aftrekken 0. Ook andere instructies hebben invloed op dit bit.
Bit 2. Dit bit wordt voor twee functies gebruikt. Ten eerste als Overflowbit "V", zoals dat beschreven is bij het rekenen met positieve en negatieve getallen. De tweede functie is die van het Pariteitscontrolebit "P". Bij het verzenden van een karakter of een commando van de computer naar een randapparaat, of andersom, van een randapparaat naar de computer, wordt gebruikgemaakt van de ASCII-code. In het algemeen is dit een zevenbits code (van 0000000 tot en met 1111111). Door storing kunnen sommige codetekens verminkt overkomen. Dit zal het ontvangende apparaat niet merken, waardoor een foutmelding uitblijft en een verkeerd karakter of commando wordt genoteerd. Om dit te ondervangen wordt aan de zeven bits van de ASCII-code nog een achtste bit toegevoegd (deze acht bits passen dan in een normaal geheugenregister). Dit achtste bit wordt voor het codegetal geplaatst en is daardoor het hoogstwaardige bit van het binaire woord geworden. Het wordt het *"pariteitsbit"* genoemd. Afhankelijk van het aantal enen in het ASCII-codeteken wordt de waarde van het pariteitsbit 1 of 0. Bij *even pariteit* wordt het pariteitsbit 1 als een oneven aantal enen in het codegetal zijn en 0 als een even aantal enen in het codegetal zijn. In dat geval is het totaal aantal enen in het woord steeds even. Het ontvangende apparaat weet dan dat er een fout bij de transmissie heeft plaatsgevonden als het een woord ontvangt met een oneven aantal enen. Bij het decoderen van het karakter of het commando zal het hoogstwaardige bit van het woord buiten beschouwing worden gelaten zodat alleen de zeven bits van het ASCII-codeteken bepalend zijn.
Oneven pariteit is ook mogelijk. In dat geval krijgt het pariteitsbit een zodanige waarde dat het totaal aantal bits in het woord oneven is.
Bij *pariteitscontrole* door de computer wordt het pariteits*controle*bit "P" ingeschakeld. Dit bit wordt 1 als een even aantal enen in een ontvangen woord zijn en 0 als een oneven aantal enen in een woord zijn. Dat houdt in dat bij even pariteit een fout is opgetreden als bit P van het FLAG-register 0 is.

Bit 3. Dit bit van het FLAG-register heeft geen bestemming en de waarde daarvan is geheel willekeurig.
Bit 4. De *Half Carry* "H" is hierin opgeslagen. Dit bit is 1 als er een overdracht is van bit drie naar bit vier in het resultaat van een berekening. Bit H wordt gebruikt voor het bepalen van een eventuele correctie bij het rekenen in de BCD-code.
Bit 5. Ook bit 5 van het FLAG-register heeft geen bestemming en zal willekeurig van waarde zijn.
Bit 6. Dit bit wordt aangeduid met "Z" van *ZERO*. Ook dit bit wordt geset of gereset, afhankelijk van het resultaat van een bewerking door de ALU in de processor. Bit Z is slechts dan 1 als alle bits in het resultaat 0 zijn. Hiermee zult u zich vast wel eens vergissen. U bent geneigd om aan te nemen dat Z de waarde 0 heeft als alle bits in het resultaat 0 zijn, maar juist het tegenovergestelde is hier het geval!
Bit 7. Bit 7 is het hoogstwaardige bit van het FLAG-register. Het wordt met "S" (Sign) aangegeven en is 1 als bit zeven van het resultaat van een bewerking door de ALU ook 1 is. Is bit zeven van het resultaat van een bewerking door de ALU 0, dan is S ook 0. Zoals beschreven in hoofdstuk 2 kan samen met het Overflowbit V worden bepaald of het resultaat een negatief dan wel een positief getal is.

Lopen we fig. 19 van boven naar beneden door, dan komen we als eerste nog niet genoemde registers de buffers W en Z tegen. Dit zijn achtbits registers die door de processor worden gebruikt om data die via de databuffers is binnengehaald, tijdelijk in op te slaan, bijvoorbeeld om een zestienbits adres tijdelijk te bewaren zodat het later in de adresbuffers kan worden geladen. Deze registers staan niet ter beschikking van de programmeur. Dat is wel het geval met de registers die hierop volgen. De registers B, C, D, E, H en L zijn alle achtbits registers. In deze registers kan data worden opgeslagen waarop direct bewerkingen kunnen worden uitgevoerd. Ook in combinatie met de Accu kan met de inhoud van de registers zeer eenvoudig worden gemanipuleerd. Omdat de registers zich in de processor bevinden behoeven ze geen nadere adressering en zijn voor de handelingen slechts één-byte instructies nodig.
Om ook zestienbits getallen in deze registers te kunnen opslaan (bijvoorbeeld adressen) kunnen steeds twee registers worden "gepaard" tot een

zestienbits register. De volgende paren kunnen worden gevormd: BC, DE en HL.

Het laatste register dat nog niet is genoemd is de *Stackpointer* SP dat in fig. 19 weer in twee delen is gesplitst, één deel voor de hoge byte (SPH) en één deel voor de lage byte. Bij de indeling van de geheugenruimte is al eerder de Stack genoemd als kladblaadje voor de computer (paragraaf 1.3.). De Stack is een last in/first out stapelgeheugen. De werking van de Stack zal met behulp van een voorbeeld worden toegelicht. Stel dat de volgende data in *deze* volgorde in de Stack moet worden bewaard: 10, 11, 12 en 13. Voor de Stack wordt een bepaald gedeelte van de geheugenruimte geadresserd. Stel dat de eerste geheugenplaats hiervoor 97FF is. De tweede geheugenplaats is dan 97FE, dus de plaats direct daaronder. De gebruikte Stackruimte groeit aan *van boven naar beneden*. Hoe de genoemde data in de Stack is opgeslagen kunt u zien in fig. 21.

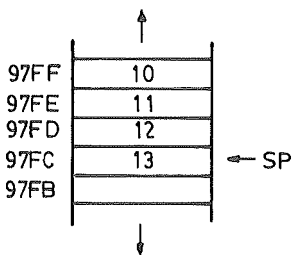


Fig. 21. De Stack.

De Stackpointer geeft steeds de geheugenplaats aan die het laatst van data is voorzien, in dit geval dus 97FC. Moet opnieuw data in de Stack worden geplaatst, dan wordt eerst de Stackpointer met 1 *verlaagd* waarna zijn inhoud in de adresbuffers wordt geladen. Nu wordt dus het adres 97FB door de adresbus aangewezen. Via de databuffers van de processor wordt deze geheugenplaats met data ingeschreven. Het uitlezen van de Stack gaat in omgekeerde volgorde. Eerst wordt de inhoud van de Stackpointer in de adresbuffers geladen. De hierdoor aangewezen geheugenplaats wordt gelezen en de data in de processor gevoerd. Daarna wordt de Stackpointer met 1 *opgehoogd*. De laatst gelezen geheugenplaats van de Stack wordt als een lege geheugenplaats beschouwd zodat de Stackpointer steeds de onderste gevulde geheugenplaats van de stapel aanwijst.

Niet alle processorregisters zijn in fig. 19 getekend. Niet getekend zijn het Interruptregister I (achtbits), het Refreshregister R (achtbits), het Index X-register IX (zestienbits) en het Index Y-register IY (zestienbits). Later zal het nut van deze registers worden verklaard. Ook is niet in de tekening aangegeven dat de registers A, F, B, C, D, E, H en L dubbel zijn uitgevoerd. Van het register A is ook een register A' in de processor, van register F een register F' enzovoorts. Er is dus een tweede set registers aanwezig. Met deze tweede (alternatieve) set kan echter niet worden gewerkt. Wel is het mogelijk de inhouden van de overeenkomstige registers uit te wisselen. Zo kan de inhoud van het register A' met dat van A worden verwisseld. Hetzelfde geldt voor de andere alternatieve registers (F' met F, B' met B enzovoorts).

3.7. Het kloksignaal.

Zoals gebleken is moet de processor nogal wat handelingen verrichten voor het uitvoeren van een enkele instructie. Dat doet hij niet zomaar vanzelf, voor elke handeling is een impuls nodig. Deze impulsen worden geleverd door een *impulsgenerator* of *klokgenerator*. De naam "klokgenerator" is ontstaan omdat de impulsen met de regelmaat van een klok na elkaar komen. Om de processor snel te laten werken is een hoge *klokfrequentie* nodig. Hiermee wordt het aantal impulsen per seconde bedoeld. Afhankelijk van het fabrikaat van de processor in uw computer kan deze een impulsfrequentie van meer dan 4 MHz. verwerken, dat wil zeggen: meer dan vier miljoen impulsen per seconde en dus ook meer dan vier miljoen handelingen per seconde. Zeker is dat uw computer sneller is naargelang de klokfrequentie hoger is. De vorm van het kloksignaal is blokvormig. Het is geschetst in fig. 22.

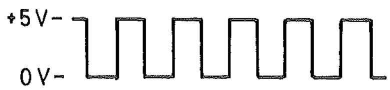


Fig. 22. Het kloksignaal.

Omdat de diverse instructies voor wat betreft hun bewerkelijkheid nogal uiteenlopen zien we ook het aantal klokimpulsen dat voor de diverse instructies nodig is uiteenlopen. Het totaal aantal impulsen per instructie wordt verdeeld in groepjes die *"machinecycli"* worden genoemd. Het kleinste aantal impulsen per machinecyclus bedraagt drie en het

grootste aantal vijf. Het aantal machinecycli dat voor een instructie nodig is kan uiteenlopen van een tot vijf (afhankelijk van de instructie). Ze zijn genummerd met M1, M2 enzovoorts. Ook de klokimpulsen binnen een machinecyclus zijn genummerd, T1, T2 enzovoorts. De eerste machinecyclus van een instructie (M1) bestaat minstens uit vier klokimpulsen (T1 tot en met T4).

3.8. De opcode-FETCH.

Bij fig. 3 is gebleken dat na het afwerken van een instructie de processor de programmateller PC met 1 moet verhogen en de nieuwe inhoud hiervan in de adresbuffers moet laden. Daarna moeten de externe stuursignalen worden gegeven voor het lezen van de geadresseerde geheugenplaats. Deze geheugenplaats zal de operatiecode bevatten van de volgende instructie. Deze operatiecode moet door middel van de databuffers worden binnengehaald en worden geplaatst in het instructieregister. Om de voortgang van het programma tot stand te kunnen brengen moet deze werkwijze, die de *"opcode-FETCH"* wordt genoemd, plaatsvinden na afloop van *elke* instructie, wat voor soort instructie dat dan ook geweest is. Bij het voorbeeld in fig. 3 is aan het eind van de eerste instructie de inhoud van de programmateller 9003. Na het voltooien van de eerste instructie (het lezen van de data 15H) begint de processor aan de volgende instructie door de inhoud van de programmateller in de adresbuffers te plaatsen. De adresbus geeft dan het adres 9003 aan en de inhoud van de desbetreffende geheugenplaats (operatiecode C6) wordt via de databuffers in het instructieregister geplaatst. Hierna begint het decoderen en worden de handelingen van de processor afhankelijk van de ingevoerde (tweede) instructie.

Deze werkwijze betekent dat de processor elke instructie met dezelfde handelwijze zal beginnen, namelijk met de opcode-FETCH. De opcode-FETCH vindt plaats gedurende de eerste drie klokimpulsen van de eerste machinecyclus van een instructie. Bij elke instructie heeft de processor nog minstens één klokimpuls nodig, vandaar dat elke eerste machinecyclus van een instructie ten minste bestaat uit vier klokimpulsen.

3.9. De wachtcyclus.

Na het doorlopen van de tweede instructie van het programma in fig. 3 staat de programmateller op 9005. Geheel overeenkomstig het voorgaande plaatst de processor deze inhoud in de adresbuffers. Hij begint dus aan een nieuwe instructie. Dat betekent dat in geheugenplaats 9005 een operatiecode moet staan waarmee de processor iets kan doen. Dit kan de operatiecode zijn van de eerste instructie van een volgend programmadeel. Het kan echter ook zijn dat het programma is beëindigd en dat er moet worden gewacht op de invoer van nieuwe gegevens via een ingangspoort, bijvoorbeeld afkomstig van een toetsenbord. De processor kan echter niet zomaar wachten. Hij begint steeds opnieuw aan een opcode-FETCH en verwacht dan een instructie die hij begrijpt. Het laten "wachten" van een computer bestaat hierin dat we hem steeds opnieuw een programmadeel laten doorlopen totdat de gewenste invoer plaatsheeft. Het programma van fig. 2 zou hiervoor moeten worden veranderd zoals in fig. 23. Voor de optelling moet het

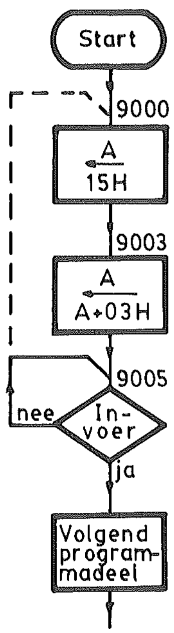


Fig. 23. De wachtcyclus.

eerste getal (15H) in de Accu worden geladen (fig. 16). De uitdrukking voor "haal 15H op uit het datageheugen" wordt daarom: "de inhoud van de Accu wordt 15H" (A ← 15H). Nu moet 03H hierbij worden opgeteld. Het resultaat vinden we weer in de Accu (fig. 17 en fig. 18). De uitdrukking hiervoor is: A ← A+03H. Tot zover is er nog niets

veranderd aan het programma. De volgende instructie moet er een zijn waarmee een ingangspoort kan worden gelezen. Deze instructie begint op het adres 9005. Er kan dan worden vastgesteld of er een invoer is geweest. Is er geen invoer geweest, dan krijgt de processor de opdracht de inhoud van de programmateller 9005 te maken. In dat geval wordt het programma vervolgd met adres 9005 en is er een sprong tot stand gekomen. Dat betekent dat opnieuw de ingangspoort wordt gelezen. Dit blijft doorgaan totdat er een invoer is geweest. Dan wordt de inhoud van de programmateller niet veranderd maar wordt gewoon doorgegaan met een volgend programmadeel. In plaats van terug te gaan naar adres 9005 hadden we ook naar het adres 9000 terug kunnen gaan. Nu wordt steeds het totale programmadeel doorlopen. Dit mag dan een vrij nutteloze handeling zijn, het steeds weer opnieuw uitrekenen van dezelfde som, maar schaden kan dat niet. De *"lus"* die we op deze manier hebben gevormd wordt een *"wachtcyclus"* genoemd.

Omdat steeds opnieuw een opcode-FETCH zal plaatshebben moet elk programma worden afgesloten met een instructie die door de computer wordt begrepen. Zo is bij het "EIND" in de figuren 4 en 6 niet alles direct maar afgelopen. Het werkelijke einde van een programma zal steeds betekenen dat de computer in een wachtcyclus terechtkomt, waarin wordt gewacht op een invoer.

Ook het systeemprogramma in uw computer kent een dergelijke wachtcyclus. Door het indrukken van een toets wordt deze wachtcyclus verlaten om een bepaalde handeling te verrichten, bijvoorbeeld het printen van het ingetoetste karakter op het scherm. Daarna wordt weer in dezelfde cyclus gewacht op de volgende invoer, tot uiteindelijk het "RUN" de processor definitief het ingevoerde programma laat uitvoeren.

3.10. De interrupt.

"Interrupt" is een uitdrukking die wordt gebruikt voor "interruptie" of "onderbreking". De bedoeling is om hiermee een lopend programma te onderbreken. Hebben we een programma zoals in fig. 5, dat maar steeds doorgaat met het doen van nutteloze handelingen, dan is het wel handig wanneer we het kunnen onderbreken zonder de computer te moeten uitschakelen. Verder is de computer zo snel dat een programma vaak wel even kan worden onderbroken voor het verrichten van een noodzakelijke handeling. De processor heeft daarvoor twee ingangen, INT en NMI. Met deze ingangen kunnen we een lopend programma onderbreken. De desbetreffende ingang, die normaal "hoog" is moet dan "laag" worden gemaakt. Het resultaat is dat de processor de instructie waarmee hij bezig is geheel afmaakt en daarna overspringt naar een bepaald programma, dat het *"interruptprogramma"* wordt genoemd. Dat wil zeggen dat in plaats van het programma waar de processor mee bezich is, het interruptprogramma wordt doorlopen. Eventueel wordt na het doorlopen van het interruptprogramma weer doorgegaan met het oorspronkelijke programma, en wel op de plaats waar het onderbroken was.

Het "laag" maken van een interruptlijn wordt in het algemeen een "interruptverzoek" genoemd (interrupt request). Aan dit verzoek hoeft de processor niet altijd te voldoen. Een interrupt kan *"gemaskeerd"* worden. Is dat gebeurd, dan wordt een interruptverzoek met de lijn INT genegeerd (niet ingewilligd). Een interruptverzoek met de lijn NMI (Non Maskable Interrupt) zal in alle gevallen worden ingewilligd. Interrupts worden bijvoorbeeld gebruikt bij het inlezen van het toetsenbord. Het lopende (systeem-)programma wordt steeds onderbroken om na te gaan of en zo ja, welke toets is ingedrukt. Op de interruptbehandeling zal nog worden teruggekomen.

4. Adresseermethoden

4.1. Inleiding.

In principe betekent "adresseren" niets anders dan het aanwijzen van een geheugenplaats van waaruit data moet worden gelezen. Deze data kan een operatiecode zijn maar ook een operand waarop de operatiecode betrekking heeft, dan wel gegevens waarmee het adres van die operand kan worden gevonden. Dit laatste houdt in dat er verschillende methoden zijn om het adres van een operand te bepalen. Deze methoden worden de *"adresseermethoden"* genoemd. Daar het adresseren bij een processor niets anders betekent dan het plaatsen van een zestienbits binair getal in de adresbuffers, betekent de adresseermethode niets anders dan de manier waarop dit getal kan worden gevonden. Hoeveel adresseermethoden een processor kent hangt onder andere af van het aantal ingebouwde registers en dat zijn er bij een Z80 processor nogal wat.

Bij het voorbeeld van fig. 3 heeft u al kennis gemaakt met twee verschillende manieren om de plaats van de operand aan te geven. Bij de eerste instructie vindt u achter de operatiecode 3A eerst de lage byte van het adres van de operand (LSB) en daarna de hoge byte van dat adres (MSB). In dit geval wordt het adres van de operand *niet* gevonden door de adresbuffers te laden met de inhoud van de programmateller. Bij de tweede instructie echter is achter de operatiecode C6 direct hierop aansluitend in het programmageheugen de operand geplaatst (03H). Nu wordt het adres van de operand dus *wel* gevonden door de adresbuffers te laden met de inhoud van de programmateller. In het voorbeeld zijn twee instructies gebruikt. De eerste was met "haal op" aangeduid. In werkelijkheid is dit de instructie "Load ACCU", afgekort met LD A. De tweede instructie "tel hierbij op" is de instructie Add ACCU, afgekort met ADD A. De hierbij toegepaste adresseermethoden behoren niet specifiek bij deze instructies. Deze adresseermethoden worden ook bij andere instructies gebruikt. Ook is het niet zo dat op de instructies alleen de gedemonstreerde methoden toegepast kunnen worden. Op de instructies kunnen nog veel meer adresseermethoden worden losgelaten.

De gebruikte afkortingen LD en ADD zeggen iets over de desbetreffende instructie, maar niet over de toegepaste adresseermethode. In werkelijkheid wordt steeds een afkorting gebruikt waaruit niet alleen de soort van instructie blijkt maar ook de gebruikte adresseermethode. De eerste instructie wordt daarom aangeduid met

LD A,(nn)

Dit betekent dat na de operatiecode nog twee bytes nn volgen. In de afkorting staan deze tussen haakjes aangegeven: (nn). Hiermee wordt aangegeven dat deze twee bytes niet zelf de operand zijn maar *het adres* van de operand vormen. Dit is in overeenstemming met wat tot heden is getoond. In code luidt de instructie:

3A 12 98

De gebruikte afkorting voor de instructie LD A,(nn) wordt het *"Mnemonic symbol"* genoemd. Vullen we in plaats van nn het adres in dan wordt de uitdrukking:

LD A,(9812H)

Het Mnemonic symbol voor de tweede instructie luidt:

ADD A,n

en met ingevulde waarde voor n:

ADD A,03H

Nu volgt achter de operatiecode slechts één byte n en omdat n nu *niet* tussen haakjes is geplaatst weten we dat hiermee de operand zelf wordt bedoeld. De instructie is in code:

C6 03

De Z80 processor kent de volgende adresseermethoden:

Immediate
Immediate Extended
Absolute
Absolute Extended
Register Indirect
Register
Register Extended
Implied
Modified Page Zero
Relative
Indexed

4.2. Immediate Addressing.

Voor wat betreft de instructieset (lijst van de instructies en hun operatiecode) zijn we in het algemeen afhankelijk van Engelstalige uitgaven. Daarom zullen de diverse adresseermethoden met hun Engelse benaming worden aangeduid. De Immediate Addressing (onmiddellijke adressering) is toegepast bij de tweede instructie van het voorbeeld in fig. 3. Hierbij is de operand *in* het programmageheugen geplaatst, direct na de operatiecode die betrekking heeft op deze operand en heeft een waarde kleiner dan 256:

Adres	Inhoud
9003	ADD
9004	03

Hierin moet voor ADD de operatiecode voor de bewerking ADD worden ingevuld.

Het Mnemonic symbool hiervoor is:

ADD A,n

en de werking:

$A \leftarrow A + n$

Deze methode wordt gebruikt als een achtbits register moet worden geladen met een getal dat onveranderlijk is (een constante), of als met een achtbits constante een bewerking moet worden toegepast.

4.3. Immediate Extended Addressing.

Zoals hiervoor is beschreven kunnen bepaalde processorregisters worden samengevoegd tot zestienbits registers. Zo kunnen de registers B en C worden samengevoegd tot een zestienbits register dat met BC wordt aangegeven. Ook kunnen de registers D en E worden samengevoegd tot het register DE en H en L tot HL. Andere zestienbits registers die ook nogal eens voor bewerkingen of voor manipulaties met getallen worden gebruikt zijn de indexregisters IX en IY en de Stackpointer. Deze manipulaties zijn mogelijk met de Immediate Extended Addressing. De desbetreffende operand volgt in het programmageheugen ook weer direct op de operatiecode die hierop betrekking heeft. De operand is echter twee bytes en er zijn daarom twee geheugenplaatsen voor elke operand nodig. De operand wordt gesplitst in een lage en een hoge byte en *in deze volgorde* in de geheugenruimte geladen. In het volgende voorbeeld wordt het register DE met het getal A54C (hex) geladen.

Adres	Inhoud
9000	LD
9001	4C
9002	A5

Het Mnemonic symbool voor deze instructie is:

LD DE,nn

Het betreft hier een twee bytes getal en daarom wordt na de komma nn gebruikt. We zullen later zien op welke manier de operatiecode voor deze LD instructie uit de instructieset kan worden bepaald. Indien een getal (of een adres) twee bytes groot is, dan wordt het steeds gesplitst in een hoge en een lage byte. Deze worden na elkaar in het geheugen geladen en steeds zo dat de lage byte ook op een lager adres staat dan de hoge byte. Het programmageheugen wordt geladen van het laagste adres naar het hoogste adres. Het getal moet dan ook in de volgorde lage byte - hoge byte naar het geheugen worden geschreven.
Omdat de hiervoor genoemde zestienbits registers op overeenkomstige wijze kunnen worden geladen is er een wat algemener Mnemonic symbool:

LD dd,nn

Hierin staat dd voor een zestienbits register BC, DE, HL of SP (Stackpointer). Betrof het een achtbits register, dan zou dat met een enkele d zijn aangegeven. De indexregisters zijn niet genoemd omdat deze een wat afwijkende operatiecode hebben.

4.4. Absolute Addressing.

De adresseermethode die voor de eerste instructie van het programma in fig. 3 is gebruikt is de Absolute Addressing mode (directe adresseermethode). Bij deze methode en ook bij de nu nog volgende adresseermethoden is er sprake van een register (bijvoorbeeld een register uit het datageheugen) dat de data bevat. Deze data kan eventueel gewijzigd worden, hetgeen niet de bedoeling is van data die in het programmageheugen is opgenomen. Het desbetreffende register is het *bronregister* (Source register). Met deze data zal in het algemeen een bewerking moeten worden uitgevoerd en zal daarom worden geplaatst in een processorregister, het *doelregister* (Destination register). Het omgekeerde is ook mogelijk: de data die het resultaat van een be-

werking is bevindt zich in een processorregister (het resultaat van een optelling bijvoorbeeld) en het kan nodig zijn om deze data in een geheugenregister op te slaan. In dat geval is het processorregister het bronregister en het geheugenregister het doelregister. In een Mnemonic symbool wordt eerst het doelregister genoemd en daarna, na een komma, het bronregister:

LD A,(nn)

Dit is het Mnemonic symbool voor de eerste instructie van ons voorbeeld. De werking is:

A ← (nn)

Het Mnemonic symbool geeft aan dat de twee bytes nn, direct volgend op de operatiecode, het adres bevatten van het bronregister. Dat deze bytes niet zelf de operand vormen, maar het adres, wordt aangegeven door de haakjes: (nn). Steeds als haakjes worden gebruikt wordt verwezen naar een adres! Het doelregister is in dit geval de Accu.

De datastroom binnen de processor is geschetst in de figuren 24 tot en met 27. Gedurende de eerste vier klokimpulsen van de eerste machinecyclus heeft de Opcode FETCH plaatsgevonden.
operatiecode 3A bevindt zich in het instructieregister (IR). Decodering hiervan heeft als resultaat gehad dat de inhoud van de programmateller PC (9001) in de adresbuffers (AB) is geladen en de data van deze geheugenplaats via de databus (DB) in de databuffers is geschreven (fig. 24). Deze data is de lage byte van het adres (12, zie fig. 3) en wordt vanuit de databuffers in het bufferregister Z gebracht. De programmateller is verhoogd tot 9002 (fig. 25). De volgende stap is dat de inhoud van de

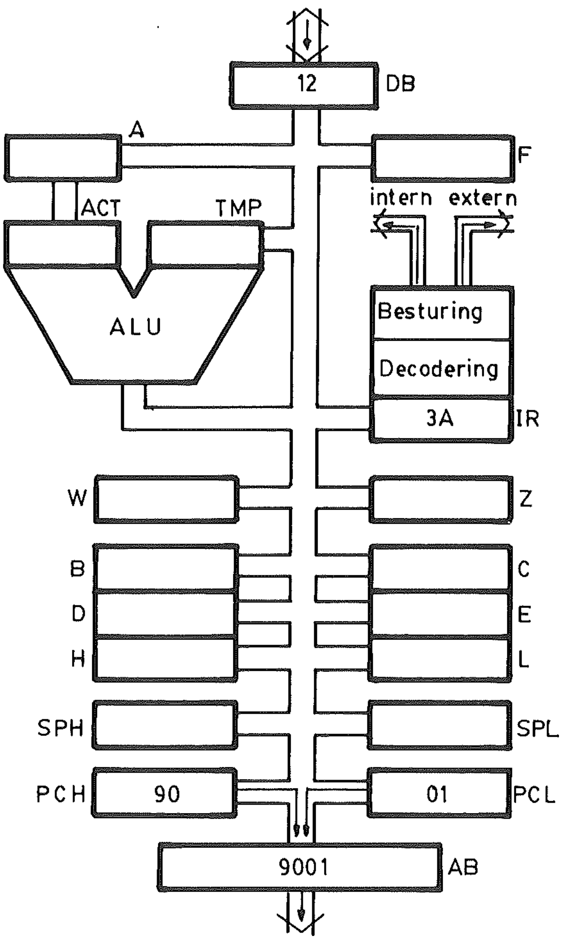


Fig. 24. Decoderen van de opcode.

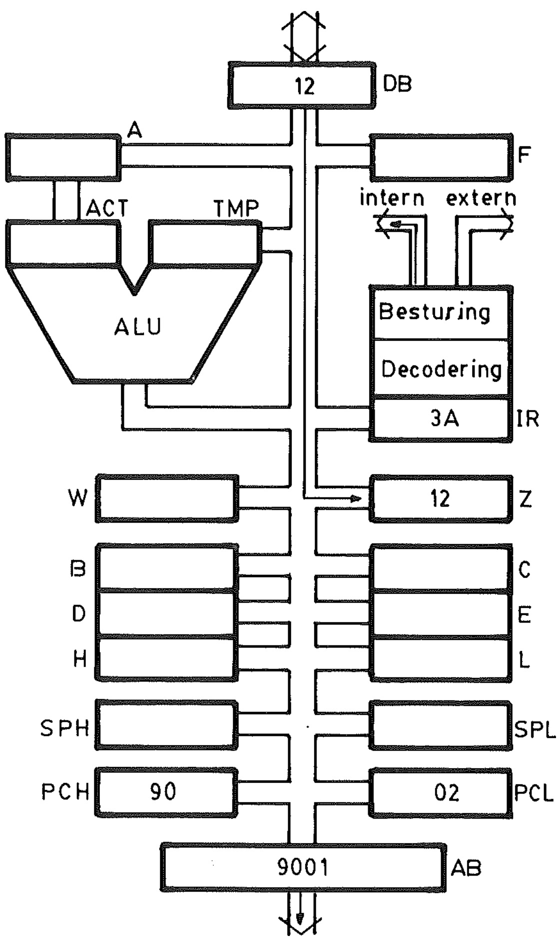


Fig. 25. Lage byte in buffer Z.

programmateller in de adresbuffers wordt geladen en nu de hoge byte van het adres (98) via de databus en de databuffers in het bufferregister W wordt gebracht (fig. 26). Als laatste deel van deze instructie wordt eerst de programmateller opgehoogd (9003) en daarna de inhoud van de registers W en Z in de adresbuffers geladen. Deze geven nu het adres 9812 aan en de data hierin (15) komt via de databus en de databuffers in de Accu (fig. 27). Voor de Opcode FETCH van de volgende instructie wordt als eerste de inhoud van de programmateller in de adresbuffers geladen. Deze geven nu het adres 9003 over aan de adresbus. Dit is het adres van de volgende instructie (operatiecode C6 voor LD A,03). Na het decoderen hiervan vindt de verdere werking plaats zoals beschreven bij de figuren 16, 17 en 18.

4.5. De Absolute Extended Addressing.
In het voorbeeld uit de vorige paragraaf is de Accu met een achtbits getal geladen (dat kan ook niet anders). Het is ook mogelijk om de Stackpointer SP en de gecombineerde registers BC, DE en HL in één instructie met een zestienbits getal te laden. Hiervoor wordt de Absolute Extended Addressing gebruikt. Een zestienbits getal is in de vorm van een hoge byte en een lage byte in twee geheugenplaatsen van het datageheugen opgeslagen. Stel dat het registerpaar BC met de data 3AC5 uit het datageheugen moet worden geladen. De instructie begint op het adres 9000 en de data bevindt zich in de geheugenplaatsen 9800 en 9801 in de volgorde lage byte - hoge byte (fig. 28). Voor de Opcode zijn, evenals voor het adres, twee bytes nodig. Dit is daarom een vier bytes instructie. Het Mnemonic

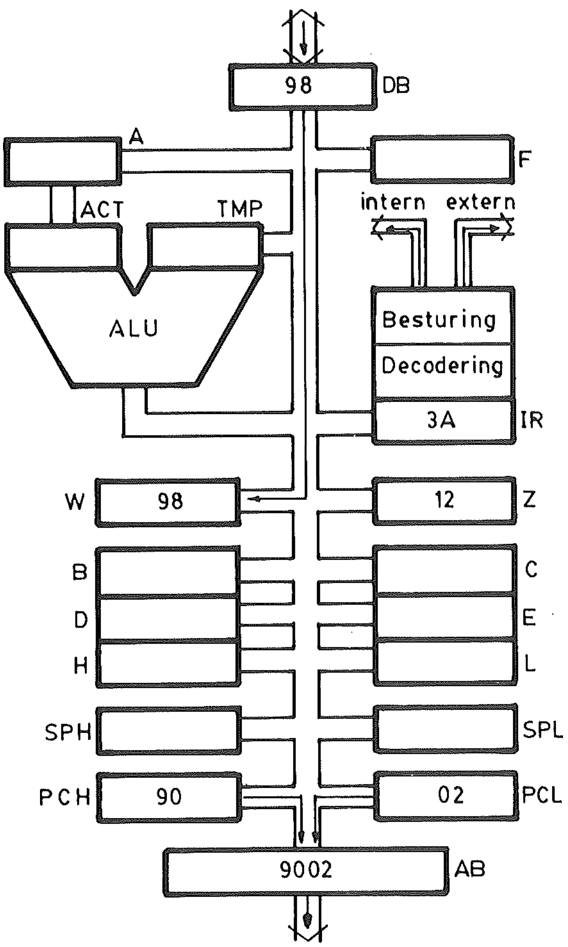


Fig. 26. Hoge byte in buffer W.

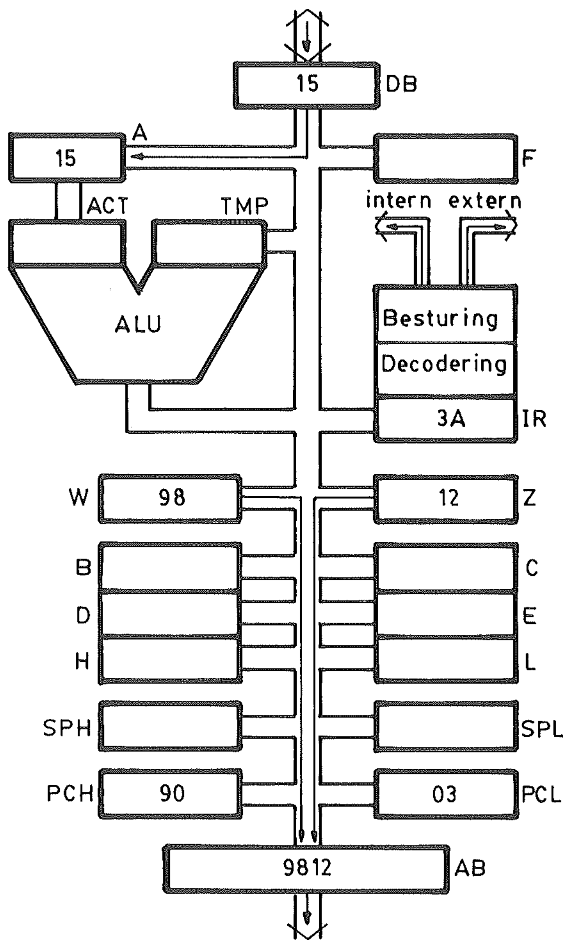


Fig. 27. Het adres van de operand in AB.

symbool voor deze instructie is:

LD BC,(nn)

In een wat algemenere vorm:

LD dd,(nn)

Hierin stelt dd het register(paar) SP, BC, DE of HL voor.

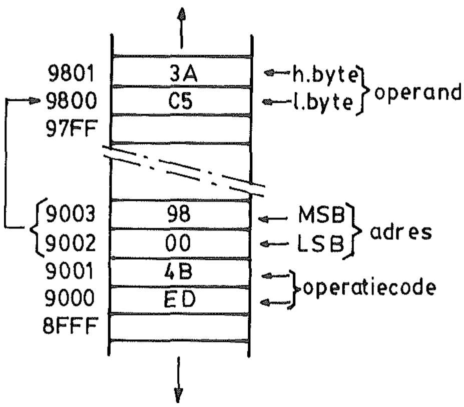


Fig. 28. De Absolute Extended Addressing.

De werking is als volgt: Na het lezen van de operatiecode wordt eerst het adres van de lage byte (9800) in de adresbuffers geladen en wordt de data (de lage byte C5) uit het geheugenregister gelezen. Deze wordt in het processorregister C geplaatst. Hierop wordt het adres in de *adresbuffers* verhoogd (9801). De tweede geheugenplaats met de hoge byte van de data (3A) wordt nu gelezen en in register B geplaatst. Daarmee is de instructie voltooid. De adressen 9800 en 9801 zijn *geen* adressen van het programmageheugen maar van het datageheugen. De programmateller heeft met deze adressen dan ook niets van doen. Na het voltooien van deze instructie heeft de programmateller 9004 als inhoud.

4.6. De Register Indirect Addressing.

In paragraaf 3.9 hebt u kennis kunnen maken met de wachtcyclus. Het zal u duidelijk zijn geworden dat een dergelijke wachtcyclus nogal eens voorkomt. Eigenlijk houdt de computer zich in veel gevallen hoofdzakelijk bezig met wachten. Zodra u de cursor op uw scherm ziet dan weet u dat de computer wacht op de invoer van een karakter via het toetsenbord. Bij veel programma's kunt u een bepaalde werking kiezen door middel van het invoeren van een bepaald karakter via het toetsenbord. Om het voorbeeld eenvoudig te houden stellen we dat u bij een tekstverwerker kunt kiezen tussen het invoeren van een tekst (bijvoorbeeld door het intoetsen van een T) en het uitprinten daarvan door middel van een printer (bijvoorbeeld door het intoetsen van een P). Het stroomdiagram geeft fig. 29.

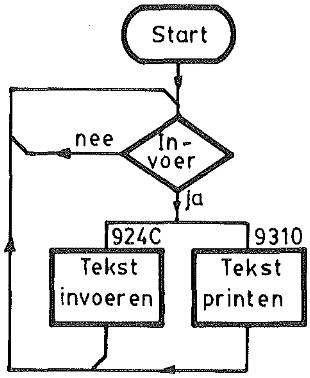


Fig. 29. Keuze uit twee programmadelen.

Na het starten van het programma komen we al gelijk in een wachtcyclus waarin wordt gewacht op de desbetreffende keuze. Is deze keuze gemaakt, dan wordt naar het desbetreffende programmadeel gesprongen. In dit schema lijkt het of beide programmadelen (invoer tekst en printen van de tekst) gelijktijdig worden ingeschakeld. Dat is natuurlijk niet waar. Deze methode van tekenen beeldt een *selectie* uit. Nu wordt geselecteerd tussen twee programmadelen, afhankelijk van de invoer. Er kan ook geselecteerd worden uit meerdere programmadelen (fig. 30). Na het invoeren van de tekst of het printen daarvan komen we weer terug bij het begin van het programma (opnieuw: wachten!). Het springen naar een gekozen programmadeel komt tot stand door een sprongopdracht: *"JUMP"*

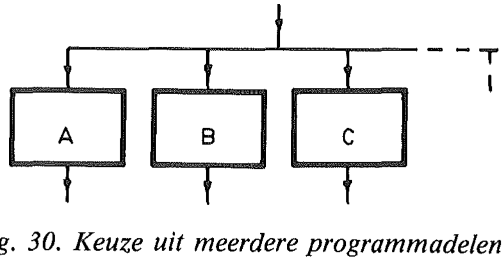


Fig. 30. Keuze uit meerdere programmadelen.

(JP). Bij een sprongopdracht wordt de programmateller PC in de processor voorzien van een nieuw adres, het adres waar naartoe moet worden gesprongen en waar het programma moet worden voortgezet.

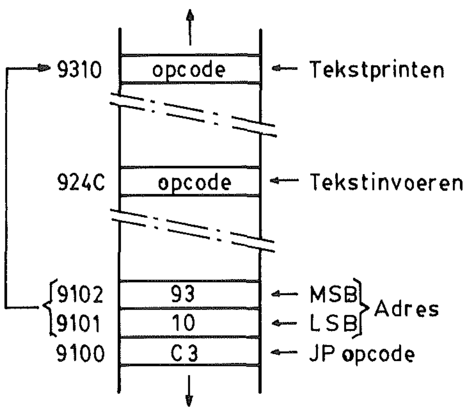


Fig. 31. Sprong naar het programmadeel "tekstprinten".

In fig. 31 begint het programmadeel voor het *invoeren* van de tekst op het adres 924C en het programmadeel voor het *printen* van de tekst op het adres 9310. Op het adres 9100 staat de opcode voor JP nn. Hierin stelt nn het sprongadres voor. In dit geval is gekozen voor het printen van de tekst zodat nn het adres 9310 aangeeft. Hiervoor moet 10 op het adres 9101 worden geplaatst en 93 op het adres 9102 (volgorde: lage byte - hoge byte). Dit is de Absolute Addressing methode. Het gevolg is dat door het uitvoeren van deze JP instructie de programmateller 9310 als inhoud krijgt en het programma op dit adres wordt voortgezet.

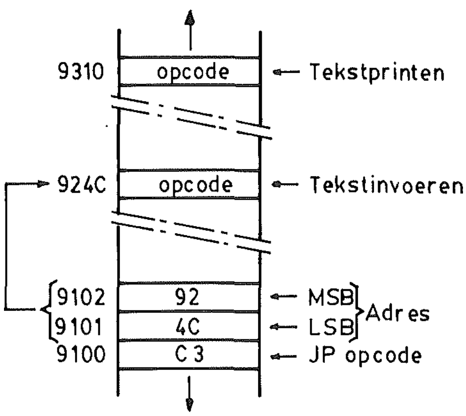


Fig. 32. Sprong naar het programmadeel "invoeren van de tekst".

Was echter gekozen voor het invoeren van de tekst dan moeten de geheugenlocaties 9101 en 9102 een andere inhoud hebben (fig. 32). Nu is al eens eerder opgemerkt dat het veranderen van de inhoud van geheugenplaatsen van het programmageheugen niet gebruikelijk is. Als het programma in een ROM-geheugenelement is opgeslagen is het veranderen van geheugenplaatsen zelfs geheel niet mogelijk. De Absolute Addressing methode kan dan in het geheel niet worden toegepast.

De *Register Indirect Addressing"* methode is hiervoor de oplossing. Daarbij wordt het sprongadres in het registerpaar HL geladen. Een enkele byte (E9) voor de opcode JP,indirect is nu genoeg om het programma de juiste sprong te laten uitvoeren (fig. 33). Als gevolg van deze opcode wordt de programmateller PC met de inhoud van het registerpaar HL geladen (fig. 34). Voor de opcode-FETCH van de volgende instructie wordt de inhoud van PC in de adresbuffers geladen en voor de daaropvolgende instructie weer met 1 verhoogd. Het programma wordt daarom vervolgd met het (nieuwe) adres dat in HL is opgeslagen.

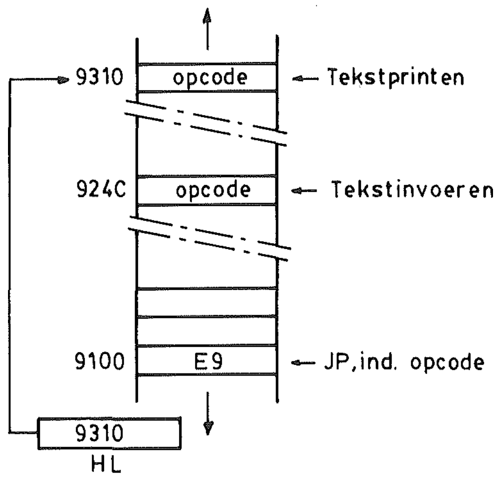


Fig. 33. De Register Indirect Addressing methode.

Is een sprong naar een ander programmadeel noodzakelijk, dan behoeft slechts de inhoud van het registerpaar HL te worden gewijzigd. Het Mnemonic symbool voor deze instructie luidt:

JP (HL)

Het betreft hier het adres dat door de inhoud van HL wordt aangegeven, vandaar de haakjes om

HL. Deze adresseermethode kan ook bij andere instructies dan JP worden toegepast (bijvoorbeeld bij LD) en kan dan ook betrekking hebben op andere registerparen in de processor (BC en DE).

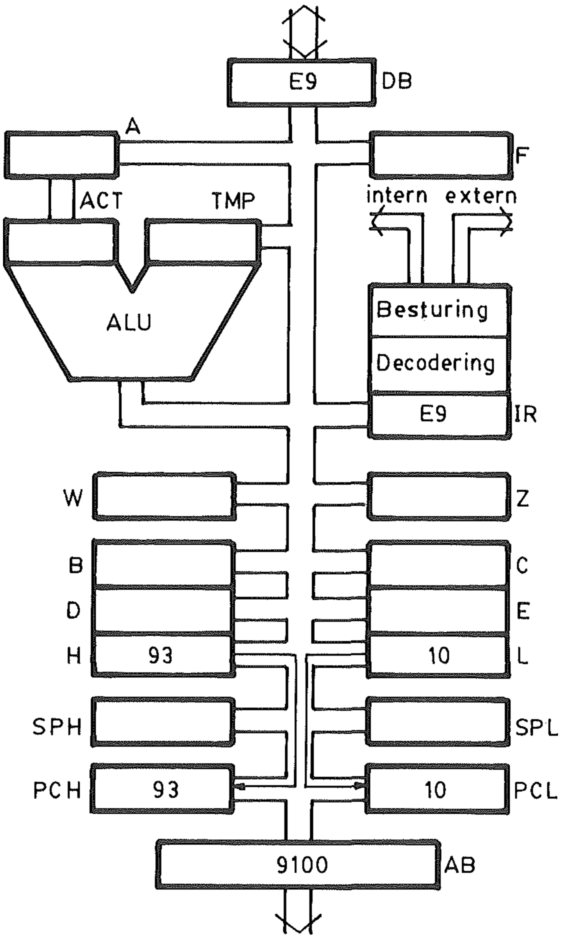


Fig. 34. PC wordt met de inhoud van HL geladen.

4.7. De Register Addressing.

Uit het voorbeeld voor de Register Indirect Addressing mode zou moeten volgen dat deze instructie slechts één byte nodig heeft terwijl de Absolute Addressing mode drie bytes lange instructies kent. Dit is een beetje bedriegelijk. Bij de Absolute Adressing mode is het adres in de instructie opgenomen terwijl bij de Register Indirect Adressing mode het desbetreffende register vooraf met het adres moet worden geladen. Dit kan met de Immediate Extended Adressing:

LD HL,nn

maar ook met de Absolute Extended Addressing:

LD HL,(nn)

Er kan echter ook met de inhouden van de processorregisters onderling worden gemanipuleerd, de Register Addressing mode. Voorbeelden hiervan zijn:

LD A,H

LD C,B

enzovoorts.

Steeds wordt weer eerst het doelregister genoemd en daarna, na de komma, het bronregister. In het eerste voorbeeld wordt de Accu met de inhoud van het H-register geladen. In het tweede voorbeeld wordt het C-register met de inhoud van het B-register geladen. Bedenk wel dat steeds de oorspronkelijke inhoud van het doelregister verloren gaat. Die van het bronregister blijft behouden.

Dit zijn manipulaties met achtbits getallen en voor de instructies is één byte genoeg. Er is ook een (beperkte) Register Extended Addressing mode voor manipulaties tussen zestienbitsregisters:

LD SP,HL
LD SP,IX
LD SP,IY

Hiervoor zijn soms twee byte instructies nodig.

4.8. Implied Addressing.

De tweede instructie van het programma van fig. 3 had het doel om het getal 03H bij de inhoud van de Accu op te tellen:

ADD A,03H

De codegetallen in het programmageheugen hiervoor zijn:

C6 03

Vaak wordt een register als een teller gebruikt. Een teller is een register waarvan de inhoud steeds met 1 wordt opgehoogd. Een teller wordt bijvoorbeeld gebruikt wanneer de processor een aantal gelijke handelingen moet verrichten. Steeds nadat een handeling is uitgevoerd wordt de teller met 1 opgehoogd zodat op deze manier wordt bijgehouden hoe vaak de handeling is uitgevoerd. Nu zou hiervoor de instructie

ADD r,01H

kunnen worden gebruikt. Hierin is r de aanduiding

van het desbetreffende register. Dit is een twee byte instructie. Omdat deze werking nogal eens voorkomt is een speciale instructie voor de Z80 beschikbaar:

INC r

INC is de afkorting voor Increment en betekent niets meer dan dat de inhoud van het register dat met r is aangegeven met 1 wordt verhoogd; r kan zijn: A, B, C, D, E, H en L.

Dit is een één byte instructie en de opcode houdt niet alleen de handeling in maar ook het doelregister (steeds een processorregister). Er wordt geen operand aangegeven in de instructie en er is ook geen sprake van een bronregister. De adresseermethode wordt *"Implied Addressing"* genoemd. Deze adresseermethode kan ook voorkomen bij bepaalde andere instructies en lijkt wel wat op Register Addressing. Bij Register Addressing is er echter steeds sprake van een bronregister en een doelregister.

Implied Addressing kan ook worden toegepast op registercombinaties (BC, DE en HL) en andere zestienbits processorregisters (bijvoorbeeld IX en IY). Men zou dan van *"Implied Extended Addressing"* kunnen spreken.

4.9. Modified Page Zero Addressing.
Bij een ander type processor, de 6502 processor, wordt de onderste pagina in het geheugen in het algemeen als datageheugen gebruikt. Deze pagina omvat de adressen 0000H tot en met 00FFH. De hoge byte van het adres (00H) geeft het paginanummer aan, hier dus pagina 00H. Om een operand op deze pagina te adresseren zouden in de Absolute Adressing mode voor het adres twee bytes nodig zijn, de hoge byte 00H en de lage byte. Bij de 6502 processor is er echter een Page Zero Addressing mode waarin slechts vermelding van het lage byte van het adres nodig is. Wordt bijvoorbeeld als lage byte 3BH aangegeven, dan wordt in de Page Zero Addressing mode de geheugenplaats 003BH geadresseerd. De Z80 processor kent deze adresseermethode niet. Slechts één instructie heeft betrekking op een adres op pagina 00H, de RST-instructie. De werking van deze instructie zal later worden behandeld. Het is een één byte instructie en de lage byte van het adres op de pagina 00H is in de operatiecode voor deze instructie verwerkt.

4.10. Relative Addressing.
In het algemeen is het wenselijk om het aantal geheugenplaatsen dat voor een programma nodig is zo klein mogelijk te houden. Dat betekent dat de instructies zo kort mogelijk dienen te zijn. De Relative Addressing mode maakt het mogelijk om bij bepaalde sprongen (JUMPS) in plaats van een drie bytes instructie een twee bytes instructie toe te passen.
Stel dat op de geheugenplaats 9100 een sprongopdracht moet komen naar het adres 9158. In de Absolute Addressing mode is hiervoor de instructie

JP nn

nodig, in codegetallen:

C3 58 91

De werking is:

PC ← 9158

hetgeen betekent dat de programmateller wordt geladen met het adres 9158. Dit is een drie bytes instructie (fig. 35).

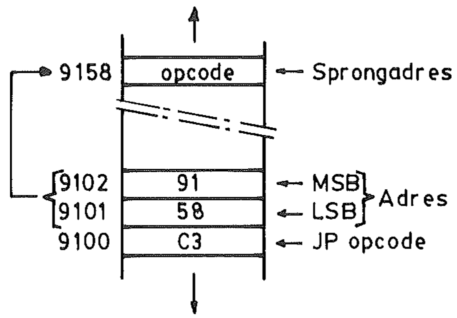


Fig. 35. Sprongopdracht in de Absolute Addressing mode.

In plaats van deze drie bytes instructie kan in dit geval ook een twee bytes instructie worden toegepast door gebruik te maken van de Relative Addressing mode. Hierbij volgt op de operatiecode een getal (één byte) dat aangeeft hoeveel geheugenplaatsen naar voren (positief) of terug (negatief) moet worden gesprongen. Een positieve sprong demonstreert fig.36. Het sprongadres is weer 9158 en de opcode (18) bevindt zich in het adres 9100. Na de opcode FETCH zal het relatieve adres 56H in de processor worden geladen. Nadat deze werking is uitgevoerd (het relatieve adres 56H is in de geheugenplaats 9101 geladen) wordt de programmateller met 1 opgehoogd en geeft dus het adres 9102 aan.

Dit zien we steeds weer terug; nadat een geheugenplaats is gelezen wordt direct de programmateller opgehoogd! Zie hiervoor ook de beschrijving bij de figuren 24 en 25. Hierna wordt het relatieve adres 56H bij de inhoud van de programmateller (9102H) opgeteld:

9102H + 56H = 9158H

De programmateller heeft nu 9158 als inhoud zodat 9158 ook het adres is waarop de volgende operatiecode in het programma zal worden gelezen.

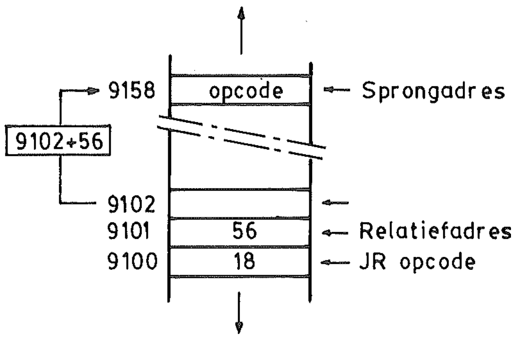


Fig. 36. Sprongopdracht in de Relative Addressing mode.

Het Mnemonic symbool voor deze instructie is:

JR e

Hierin is JR de afkorting voor JUMP Relative en is e het getal dat bij de inhoud van de programmateller moet worden opgeteld om het sprongadres te vinden. Dat getal wordt het *"relatieve adres"* genoemd. We moeten daarbij steeds bedenken dat de inhoud van de programmateller, waarbij het relatieve adres wordt opgeteld, steeds twee hoger is dan het adres waarop zich de opcode bevindt! De werking van de instructie is te noteren als:

PC ← PC + e

De relatieve sprong in fig. 36 is positief (vooruit, naar een hoger adres). Hoe ook een negatieve sprong kan worden gemaakt zien we bij de behandeling van de spronginstructies.

4.11. Indexed Addressing.

Als laatste van de adresseermethoden volgt hier de Indexed Addressing mode. Bij deze methode wordt gebruikgemaakt van het Index-X register (IX) of het index-Y register (IY). De Indexed Addressing wordt veel gebruikt voor het opzoeken van een bepaald getal uit een tabel. Tabellen komen vaak voor bij programma's en kunnen zowel reeksen van getallen zijn als reeksen van codegetallen die karakters voorstellen, zoals dat bijvoorbeeld bij de ASCII-code het geval is. Het eerste getal van de reeks is dan geladen in een geheugenplaats die wel het *"basisadres"* van de tabel wordt genoemd. Alle volgende getallen zijn in *volgorde* in de hierop aansluitende reeks geheugenplaatsen geladen. Stel dat we uit een tabel een bepaald getal willen opzoeken met behulp van het IX register. Het basisadres van de tabel is 9850 en het desbetreffende getal moet uit de tabel in de Accu worden geladen. De operatiecode voor de instructie hiervoor is twee bytes groot: DD 77. Volgend op deze operatiecode volgt de lage byte van het basisadres van de tabel: 50. De volledige operatiecode voor deze instructie is daarom:

Adres	Inhoud
9100	DD
9101	77
9102	50

Aangenomen is dat de instructie begint op het adres 9100. Het index-X register is een zestienbits register en kan daarom een hoge en een lage byte bevatten. De hoge byte in dit register is de hoge byte van het basisadres: 98. De lage byte in het IX register geeft de plaats in de tabel aan. Om de eerste plaats (plaats 0) van de tabel te adresseren moet de inhoud van het index-X register gelijk zijn aan 9800, voor de volgende plaats 9801 enzovoorts. Het adres van het desbetreffende getal in de tabel wordt gevonden door de derde byte van de instructie (50) op te tellen bij de inhoud van het index-X register. In fig. 37 is getoond hoe het adres van het negende getal in de tabel (plaatsnummer 8) wordt berekend. De lage byte in het IX register is 08 zodat de inhoud van het register 9808 is. Hierbij wordt de LSB van het basisadres opgeteld: 9808H + 50H = 9858H.

Door deze instructie steeds te herhalen, waarbij de inhoud van het IX register bij elke herhaling met 1 wordt opgehoogd, kan de gehele tabel stap voor stap worden doorlopen.

Het Mnemonic symbool voor de instructie is:

LD A,(IX + d)

Hierin is d de lage byte van het basisadres.

Wat algemener luidt de instructie:

LD r,(IX + d)

Hierin kan r het register A, B, C, D, E, H of L zijn. Dezelfde werking is met het index-Y register (IY) mogelijk:

LD r,(IY + d)

De Indexed Addressing mode kan ook nog op andere instructies dan LD worden toegepast.

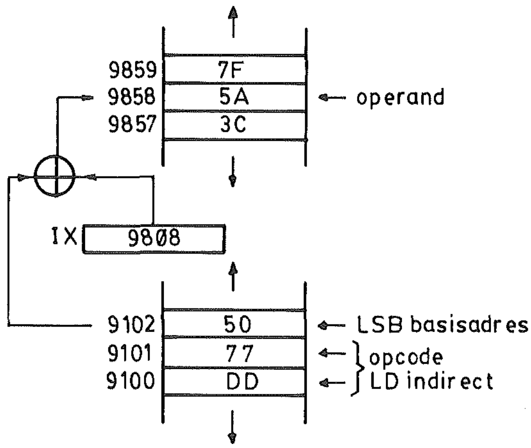


Fig. 37. Indexed Addressing mode.

5. In- uitvoer en interrupts

5.1. In- en uitvoer.
In paragraaf 1.1 is al vermeld dat de in- en de uitvoer van gegevens van en naar de randapparatuur loopt via de in- en de uitgangspoorten. Van de processor uit gezien zijn de ingangspoorten niets anders dan registers die kunnen worden uitgelezen zoals dat met geheugenregisters het geval is. Dat tussen het register en het randapparaat nog uitgebreide elektronische schakelingen kunnen zijn aangebracht, daar merkt de processor in principe niets van. Zijn taak is het slechts om de informatie die op de ingangspoorten ter beschikking staat op een bepaald moment te lezen. Hetzelfde geldt voor de uitgangspoorten. Hoewel dit in bepaalde gevallen *buffers* kunnen zijn (elektronische schakelingen die een bepaald elektrisch vermogen kunnen leveren aan de hierop aangesloten schakelingen), de processor ziet ze slechts als registers die met informatie kunnen worden ingeschreven. De in- en de uitgangspoorten zouden dan ook gewoon in de geheugenruimte opgenomen kunnen worden en van een bepaald adres kunnen worden voorzien. Dit is echter niet gebeurd. De in- en de uitgangspoorten zijn samengevoegd tot een zelfstandig blok registers, afzonderlijk van de geheugenelementen. De besturingslijnen van de processor kunnen de geheugenelementen en de in- en de uitgangsregisters (poorten) afzonderlijk van elkaar activeren. Er zijn daarom aparte instructies voor het lezen van de geheugenelementen en voor het lezen van de ingangspoorten. Hetzelfde geldt voor het schrijven naar de geheugenelementen en naar de uitgangspoorten.

Ook de diverse poorten moeten een nummer (een adres) toegewezen krijgen. Er zijn maximaal 256 (2^8) ingangspoorten mogelijk en deze zijn genummerd van 00H tot en met FFH. Voor het adresseren worden de adreslijnen A0 tot en met A7 gebruikt, de adreslijnen die de lage byte van het adres bevatten. Deze poorten zijn slechts uit te lezen. Omdat de uitgangspoorten alleen maar zijn in te schrijven en ook weer afzonderlijk van de ingangspoorten worden geactiveerd, kunnen deze dezelfde adressen hebben als de ingangspoorten, dus ook van 00H tot en met FFH. Elk poortregister bevat acht poorten voor acht in- of uitgangslijnen, zodat voor de data-uitwisseling tussen de processor en de poortregisters de databus kan worden gebruikt. Fig. 38 schetst de organisatie van de poorten.

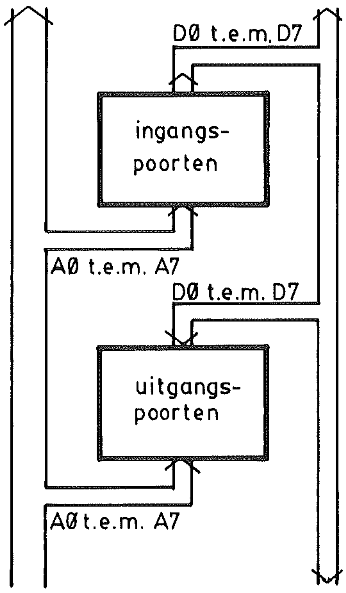


Fig. 38

De situatie voor een enkel ingangsregister voor acht ingangspoorten geeft figuur 39. Behalve de acht adreslijnen en de acht lijnen van de databus

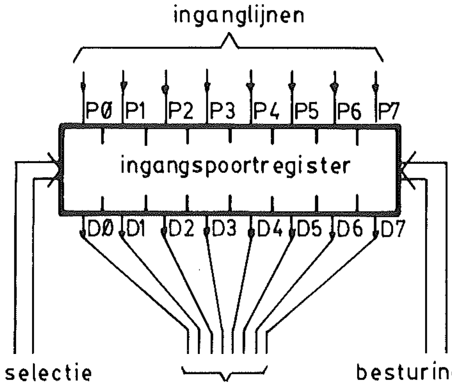


Fig. 39

zijn er ook nog besturingslijnen op aangesloten die bepalen of een register actief moet zijn. Verder zijn er de acht ingangslijnen die, eventueel via elektronische schakelingen, de verbinding verzorgen van het randapparaat met de computer. De situatie voor een uitgangsregister is hiermee overeenkomend.

5.2. Interrupts, algemeen.

Data-uitwisseling tussen de poorten en de processor is slechts mogelijk door middel van instructies die in een programma zijn opgenomen. Dat zou betekenen dat voor het invoeren van gegevens door middel van het toetsenbord moet worden gewacht tot het programma dat mogelijk maakt. Het kan echter wel eens nodig zijn dat de invoer dient te geschieden op een willekeurig tijdstip, onafhankelijk van het programma. Daartoe kan het lopende programma worden onderbroken met een interrupt. Treedt een interrupt op dan wordt na het onderbreken van het "hoofdprogramma" een specifiek programmadeel doorlopen, bijvoorbeeld een programmadeel voor het lezen van het toetsenbord. Een dergelijk programmadeel wordt wel een *"interruptroutine"* of *"interruptprogramma"* genoemd. Na het "afwerken" van de interruptroutine wordt het hoofdprogramma vervolgd op de plaats waar het was onderbroken. De mogelijkheid van een interrupt is al aangegeven in paragraaf 3.10.

Een interrupt komt "hardware" tot stand, dat wil zeggen dat op een bepaalde ingangslijn van de processor een impuls wordt gegeven. Dit houdt in dat de desbetreffende lijn voor enige tijd "laag" gemaakt wordt. Er zijn twee ingangslijnen die een interrupt tot stand kunnen brengen, de NMI lijn en de INT lijn. Het laag maken van de NMI lijn of de INT lijn wordt tot stand gebracht door het randapparaat dat een interrupt wil veroorzaken.

In principe is het zo dat een randapparaat een interrupt aanvraagt bij de processor door het laag maken van de desbetreffende lijn. De processor kan deze *"interrupt request"* inwilligen door een interrupt te veroorzaken, dat wil zeggen: het lopende programma onderbreken maar niet nadat de instructie waarmee de processor bezig was geheel is afgewerkt. De processor kan in bepaalde gevallen ook de interruptaanvraag negeren. Wordt de NMI lijn laag gemaakt, dan zal de processor altijd een interrupt veroorzaken. De afkorting NMI is afkomstig van: Non Maskable Interrupt. Naast deze niet maskeerbare interrupt is er ook een maskeerbare interrupt, aangegeven met INT. Hiervoor dient de tweede lijn en het laag maken hiervan heeft niet altijd een interrupt tot gevolg.

Een register in de MSX computer is een samenstelling van acht gelijkwaardige elektronische schakelingen en elke schakeling kan een bit bevatten van een binair getal. Er zijn schakelingen die kunnen worden ingeschreven met een 1 of een 0. Zij hebben dan het vermogen om twee standen in te nemen, net zoals een wip wap uiteindelijk in één van zijn twee uiterste standen blijft staan. Een dergelijke schakeling wordt een flip flop schakeling genoemd en een achtbits register bestaat uit acht flip flop schakelingen. In de processor zijn een aantal zelfstandige flip flop schakelingen aanwezig die het doel hebben om een bepaalde situatie te onthouden. Het inschrijven met een 1 wordt het *"setten"* van de flip flop genoemd en het inschrijven met een 0 het *"resetten"*. Om te kunnen onthouden of er een interrupt aanvraag is geweest zijn er in de processor twee flip flop schakelingen, de NMI flip flop (NMI FF) en de INT flip flop (INT FF). Normaal zijn deze gereset (0) maar door het laag maken van de desbetreffende lijnen worden ze geset. Verder is er een flip flop waaraan de processor kan zien of een INT aanvraag moet worden ingewilligd of moet worden genegeerd. Dit is het Interrupt flip flop IFF1. Is dit gereset (0), dan wordt de INT aanvraag genegeerd; is het geset, dan wordt de aanvraag ingewilligd. Deze flip flop wordt "software" geset of gereset door middel van een instructie in een programma.

Het kan zijn dat twee randapparaten gelijktijdig of bijna gelijktijdig een INT en een NMI aanvragen. Gebeurt dit tijdens het afwerken van een instructie door de processor, dan kan deze nog op geen der beide aanvragen reageren. Na het afwerken van de instructie wordt eerst op de NMI aanvraag gereageerd en daarna pas op de INT aanvraag. Aan de NMI aanvraag wordt een *hogere prioriteit* gegeven dan aan een INT aanvraag. De verwerking van een interrupt verloopt zoals in fig. 40 met een stroomdiagram is aangegeven.

Afhankelijk van de lijn waarlangs de impuls binnenkomt wordt de NMI FF of de INT FF geset. Als beide lijnen geactiveerd zijn, dan worden ook

beide flip floppen geset. Hiermee is vastgelegd welke interrupt is aangevraagd en hebben de INT en NMI ingangslijnen van de processor geen nut meer. Nu wordt nagegaan of de lopende instructie al is beëindigd. Is dat niet het geval, dan wordt hij eerst afgemaakt. Hierna wordt nagegaan of de NMI FF is geset. Is dat het geval, dan wordt naar de NMI interruptroutine gesprongen. Dit is een subroutine. Onder een subroutine bij machinetaal programma's kunt u ongeveer hetzelfde verstaan als die bij een BASIC programma. Na het doorlopen van de routine wordt weer teruggesprongen naar de instructie in het programma volgend op die waardoor de sprong tot stand kwam. Daarom wordt na het doorlopen van de NMI interruptroutine ook nagegaan of de INT FF is geset. Is dat het geval, dan wordt IFF1 getest. Heeft deze de waarde 1 dan is de interrupt toegestaan zodat naar de INT routine wordt gesprongen. Heeft IFF1 de waarde 0, dan wordt gewoon doorgegaan met de volgende instructie van het programma.
Uit het verloop van het stroomdiagram in fig. 40 volgt dat bij gelijktijdige aanvraag van een NMI en een INT de NMI het eerst zal worden uitgevoerd.

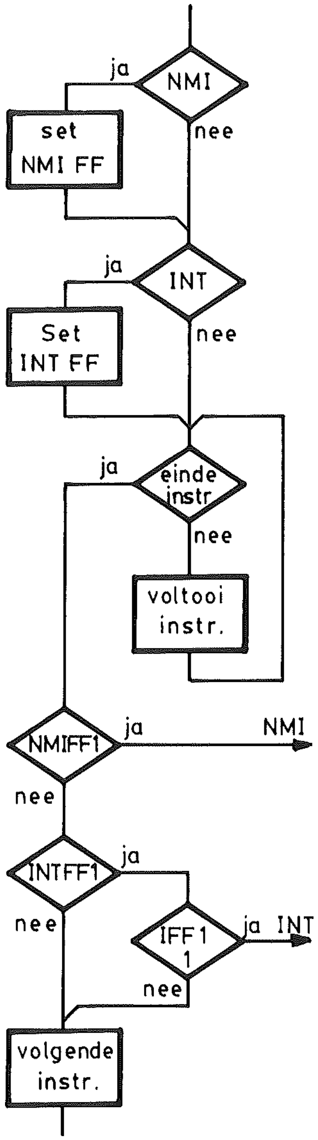


Fig. 40. De interrupt behandeling.

5.3. De niet-maskeerbare interrupt.

Voordat een NMI interruptroutine kan worden doorlopen zullen enige voorzorgsmaatregelen moeten worden genomen. Ten eerste is de interruptroutine een subroutine zodat het adres van de eerstvolgende instructie van het programma dat werd onderbroken, zal moeten worden bewaard. Dat is nodig om later het programma op dit adres weer te kunnen voortzetten. Van dit programma is zojuist een instructie voltooid en zoals al eens eerder is gedemonstreerd bevat de programmateller dan het adres van de volgende instructie. Dit adres zal nu moeten worden bewaard. Nu komt het nut van de Stack naar voren. De inhoud van de programmateller wordt naar de Stack geschreven. Deze handeling wordt geheel automatisch door de processor verricht aan het begin van de interruptbehandeling. Stel dat de Stackpointer SP het adres 9800 aangeeft (de Stackpointer kan door middel van een instructie van een bepaalde inhoud worden voorzien). Deze wordt nu allereerst met 1 verlaagd tot 97FF. Op dit adres wordt de inhoud van het hoge deel van de programmateller (PCH) geschreven. Nu wordt de Stackpointer weer met 1 verlaagd en geeft het adres 97FE aan. Op dit adres wordt de inhoud van het lage deel van de programmateller (PCL) geladen (fig. 41). De Stackpointer behoudt de waarde 97FE, zodat steeds kan worden vast-

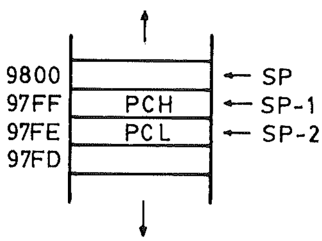


Fig. 41. Bewaren van het retouradres.

gesteld waar het adres is opgeslagen waarmee uiteindelijk het oorspronkelijke programma moet worden voortgezet.

Nu zou de programmateller met het adres van de interruptroutine kunnen worden geladen. Eerst is echter nog een andere handeling nodig. De interruptroutine is een normaal programmadeel en tijdens het doorlopen van dit programmadeel zou door een of ander randapparaat een INT kunnen worden aangevraagd. Uiteraard moet eerst de lopende interruptroutine worden voltooid zodat de INT niet mag worden gehonoreerd. Daarvoor moet IFF1 0 worden gemaakt, nadat de oorspronkelijke inhoud hiervan in een aparte flip flop (IFF2) is opgeslagen, dus: De inhoud van IFF1 wordt in IFF2 opgeslagen en IFF1 wordt 0. Nu kan geen INT meer plaatshebben.

De volgende handeling betreft het laden van PC met het beginadres van de interruptroutine. Dit gebeurt met de indirecte adresseermethode, een adresseermethode die in het algemeen bij de Z80 niet apart wordt genoemd omdat deze slechts bij de NMI voorkomt. Op het adres 0066 bevindt zich de lage byte van het adres van de interruptroutine en op 0067 de hoge byte hiervan. De processor leest de waarden en laadt deze in de programmateller. Nu het startadres van de interruptroutine bekend is zal deze vanaf dit adres worden doorlopen. Aan het einde van deze routine moet een instructie voorkomen die aangeeft dat weer terug moet worden gegaan naar het onderbroken programma. Dit is een overeenkomstige instructie als het BASIC statement RETURN en is voor de processor het bevel dat een en ander weer in de oorspronkelijke situatie moet worden teruggebracht. Daarvoor wordt eerst de inhoud van IFF2 in IFF1 geladen, zodat deze laatste weer de oorspronkelijke inhoud heeft van vóór de interrupt behandeling. Hiermee is het weer bekend of een eventuele hiernavolgende INT al of niet kan worden uitgevoerd. Daarna moet het retouradres uit de Stack worden opgehaald. De inhoud van SP is nog steeds 97FE en deze waarde wordt in de adresbuffers geladen. Op dit adres staat de lage byte van het retouradres en dit wordt in PCL geladen (fig. 40). De Stackpointer wordt met 1 opgehoogd en geeft daardoor het adres 97FF aan. Hier vinden we de hoge byte van het retouradres dat in PCH wordt gebracht. Als laatste handeling wordt de Stackpointer nogmaals met 1 verhoogd zodat deze weer de oorspronkelijke inhoud 9800 heeft. Het geheel van handelen is voorgesteld in fig. 42.

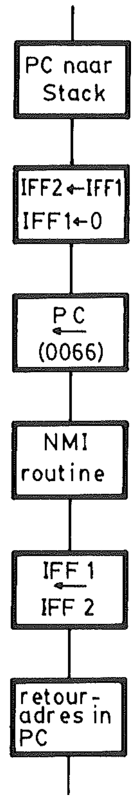


Fig. 42. De NMI behandeling.

5.4. De INT, mode 0.

In tegenstelling tot de NMI is de INT *"maskeerbaar"*; dat betekent dat door het 0 maken van IFF1 (resetten) een eventueel interruptverzoek met de INT lijn niet zal worden ingewilligd. Het setten of het resetten van IFF1 wordt door middel van instructie in een programma tot stand gebracht. Er is maar één manier waarop een maskeerbaar interrupt kan worden veroorzaakt, namelijk een impuls over de INT lijn. Er zijn echter drie manieren om op de interrupt te reageren: mode 0, mode 1 en mode 2. Eén van deze moden kan worden gekozen door middel van een instructie in een programma. Na het inschakelen van de computer wordt in eerste instantie door de processor zelf de mode 0 gekozen. Om de juiste werking van deze mode te kunnen begrijpen is nogal wat kennis van de hardware organisatie van de computer nodig. In principe komt het er op neer dat de processor, na het on-

derbreken van het lopende programma, blijft wachten tot hem een instructie via de databus wordt aangeboden. Deze instructie komt dan niet uit het programmageheugen maar wordt door het randapparaat dat de interrupt heeft aangevraagd, op de databus geplaatst. De instructie kan een RST instructie zijn. Deze is al eens genoemd bij de behandeling van de Modified Page Zero Addressing (paragraaf 4.9.). In dit geval wordt een routine op een adres van pagina 00H afgewerkt.
Een tweede mogelijkheid is dat het randapparaat een spronginstructie naar een subroutine op de databus plaatst. In dat geval moet deze instructie gevolgd worden door een tweebytes adres dat eveneens door het randapparaat op de databus moet worden geplaatst. Besturingslijnen tussen de processor en het randapparaat moeten er voor zorgen dat een en ander vlekkeloos verloopt.
Het blokschema van de interruptbehandeling geeft fig. 43.

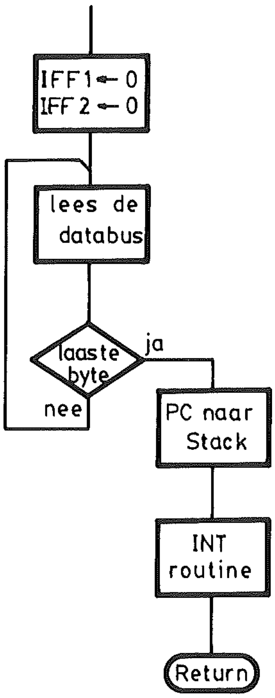


Fig. 43. Maskeerbare interrupt, mode 0.

Direct na het beëindigen van de instructie van het programma waarin de interruptaanvraag is binnengekomen worden IFF1 en IFF2 gereset. Omdat IFF1 daardoor 0 is geworden zijn geen verdere interrupts meer mogelijk. Deze situatie wordt niet meer automatisch door de processor veranderd. Dat betekent dat de programmeur door middel van een instructie, IFF1 moet setten als het verdere verloop van het programma dat nodig maakt. Omdat de interrupt op elke willekeurige plaats in een programma kan plaatshebben gebeurt het setten van IFF1 vaak aan het einde van het interruptprogramma. Na het resetten van IFF1 en IFF2 wordt op de instructie gewacht die op de databus moet worden geplaatst. Is de instructie (plus het eventuele adres) door de processor gelezen, dan wordt het retouradres, dat zich nog in de programmateller bevindt, naar de Stack geschreven, zodat na het doorlopen van de interruptroutine de processor het onderbroken programma kan vervolgen waar het was gestopt. Hierna volgt het afwerken van het interruptprogramma, dat met een zodanige instructie is afgesloten dat weer met het oorspronkelijke programma kan worden doorgegaan. Hiertoe wordt het retouradres weer uit de Stack gelezen. De Stack-handelingen verlopen geheel zoals bij fig. 41 is beschreven. In principe is de gehele werking gelijk aan die welke wordt gevolgd bij het aanroepen van een subroutine.

5.5. De INT, mode 1.

De maskeerbare interrupt mode 1 vraagt heel wat minder "hardware" dan die van mode 0. Bij mode 0 is in het randapparaat of eventueel in de computer zelf de nodige logica nodig, meestal in de vorm van IC's, om de instructie en het eventuele adres van het interruptprogramma op de databus te plaatsen. Bij de mode 1 zal de processor echter steeds na het resetten van IFF1 en IFF2 en het bewaren van het retouradres in de Stack, het adres 0038 in PC laden. Op dit adres moet de eerste instructie staan van de interruptroutine. Ook deze in-

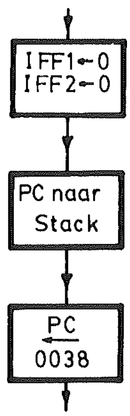


Fig. 44. Maskeerbare interrupt, mode 1.

terruptroutine kan worden beschouwd als een subroutine en moet als laatste instructie dus een RETURN instructie bevatten. Hierdoor wordt het returnadres weer uit de Stack gelezen zodat het oorspronkelijke programma kan worden voortgezet op de plaats waar het was verlaten. Ook nu is het aan de programmeur overgelaten om voor een eventuele volgende interrupt, IFF1 te setten. Het blokschema van de interruptbehandeling geeft fig. 44.

5.6. De INT, mode 2.

In paragraaf 3.6. is de aanwezigheid gemeld van het Interruptregister I in de processor. Tot nu toe zal u het nut van dit register ontgaan zijn. Bij de maskeerbare interrupt mode 2 speelt dit register echter een belangrijke rol. Het is een acht bits register en de inhoud hiervan wordt gebruikt als de hoge byte van een indirect adres. De lage byte van dit indirecte adres wordt door het randapparaat dat de interrupt heeft aangevraagd, op de databus geplaatst. Het indirecte adres is een geheugenplaats van het datageheugen en bevat het uiteindelijke adres van de interruptroutine. Het is op deze manier mogelijk dat het randapparaat kiest uit verschillende interruptroutines of dat verschillende randapparaten elk hun eigen interruptroutine in werking stellen.

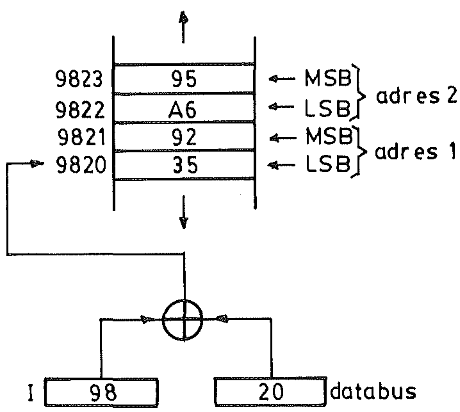


Fig. 45. De indirecte adressering van de interruptroutine.

In fig. 45 begint een tabel van adressen van interruptroutines op het adres 9820. Op dit adres staat de lage byte van een adres en op de volgende geheugenplaats de hoge byte van dat adres. In dit voorbeeld is het eerste adres in de tabel 9235 en het volgende adres 95A6. De tabel *moet* steeds beginnen op een *even* adres. Ook in dit geval is het beginadres van de tabel een even getal (9820). Het gevolg is dat de lage bytes van de adressen van de interruptroutines steeds worden gevonden op een even indirect adres. De hoge byte van het indirecte adres moet in het Interruptregister geladen zijn. Het I register bevat in dit voorbeeld dus het getal 98. Het randapparaat brengt de lage byte van het indirecte adres op de databus, hier 20. Plaatst het randapparaat het getal 21 op de databus, dan verandert de processor zelf dit in 20. Hiermee wordt het indirecte adres 9820 gevormd. De processor leest op dit adres de lage byte 35 en op de volgende geheugenplaats de hoge byte 92 en heeft daarmee het adres van de gewenste interruptroutine gevonden.

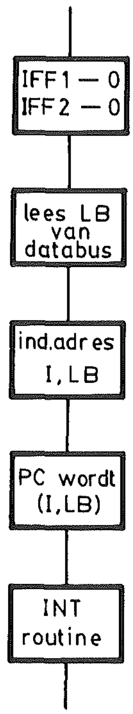


Fig. 46. Maskeerbare interrupt, mode 2.

De verdere interruptbehandeling is gelijk aan die van mode 0 en mode 1 (fig. 46).
Eerst worden IFF1 en IFF2 gereset zodat een volgende interrupt niet mogelijk is. In een programma moet IFF1 eventueel later weer worden geset. Daarna wordt de inhoud van de programmateller naar de Stack geschreven. Dan volgen de handelingen die bij fig. 45 zijn beschreven en kan de interruptroutine worden doorlopen.

6. De instructieset

6.1. Inleiding.

In het algemeen kan de *"instructieset"* van een bepaalde processor omschreven worden als de verzameling van instructies die door de processor kunnen worden uitgevoerd. Meestal wordt de instructieset beschouwd als een lijst waarin deze instructies zijn beschreven. Om een programma te kunnen maken moet deze lijst de volgende gegevens bevatten:

a. De *mnemonic symbolen* van de instructies.
b. Bij elke instructie de beschrijving van de werking.
c. Welke adresseringsmethode(n) op de desbetreffende instructie kunnen worden toegepast.
d. De *operatiecode* voor elke instructie, in samenhang met de toe te passen adresseringsmethoden.
e. Welke *FLAGS* in het *FLAG-register* door de desbetreffende instructie worden geactiveerd en op welke wijze deze worden geactiveerd (setten, resetten of veranderen).
f. Het aantal klokimpulsen dat de processor nodig heeft om de instructie uit te voeren (eventueel ook het aantal machinecycli).
g. Het totaal aantal bytes (voor de operatiecode en het adres) van de instructie.

Omdat verschillende instructies dezelfde of ongeveer dezelfde werking hebben is de instructieset ingedeeld in groepen. Deze indeling is specifiek voor een bepaalde processor en kan daarom per processor verschillen. De bedoeling van dit hoofdstuk is om de instructies per groep te beschrijven. De bij de Z80 gebruikelijke indeling is:

a. Datatransportinstructies (load en exchange group).
b. Rekenkundige en logische instructies (arithmetic and logical group).
c. Rotatie en schuifinstructies (rotate en shift group).
d. Bitmanipulatie-instructies (bit manipulation group).
e. Spronginstructies (jump, call and return group).
f. Invoer- en uitvoerinstructies (input/output group).
g. Controle instructies (CPU control group).
h. Bloktransportinstructies (Block transfer and search group).

De operatiecoden van de instructies zijn in veel instructiesets voor de Z80 CPU in één of meer woorden van achtbits gegeven. Dat komt omdat u hierin zelf nog een aantal bits een waarde moet geven, afhankelijk van het register waarop de instructie van toepassing is.

6.2. Datatransportinstructies.

De datatransportinstructies hebben betrekking op het uitwisselen van gegevens (data) tussen diverse registers. Dit kunnen zowel registers binnen de processor zijn als registers in het geheugen. Is bij de uitwisseling een geheugenregister betrokken, dan is het tweede register steeds een processorregister. Door middel van een enkele instructie is het dus niet mogelijk om data-uitwisseling tot stand te brengen tussen twee geheugenregisters. Dit laatste dient via de processor te geschieden en er zijn daarvoor steeds ten minste twee instructies nodig, één om de desbetreffende data vanuit een geheugenregister in de processor te laden en daarna nog één om deze data vanuit de processor in het gewenste geheugenregister te brengen.

De datatransportinstructies zijn onder te verdelen in vier groepen:
a. De achtbits LOAD-instructies.
b. De zestienbits LOAD-instructies.
c. De STACK-instructies.
d. De dataverwisselinstructies (EXCHANGES).

a. De achtbits LOAD-instructies.

Bij de achtbits LOAD-instructies is steeds sprake van een *bronregister* (SOURCE) en een doelregister (destination). De werking van de instructie is steeds zo dat de inhoud van het bronregister in het doelregister wordt geplaatst. De oorspronkelijke inhoud van het doelregister gaat daardoor verloren, dat van het bronregister blijft echter behou-

den. Dat betekent dat indien na elkaar een aantal registers worden geladen vanuit hetzelfde bronregister, al deze registers dezelfde inhoud krijgen.
Er zijn een groot aantal achtbits LOAD-instructies. De reden hiervan is dat er een groot aantal adresseermogelijkheden bij deze groep toegepast kunnen worden. Om te beginnen de register addressing mode (paragraaf 4.7).
De instructie hiervoor is

LD r,s

Laad processorregister r met de inhoud van processorregister s.

Hierbij is zowel r als s het register A, B, C, D, E, H of L van de processor.
Is het doel het register B en de bron het register H, dan luidt de instructie:

LD B,H

Laad register B met de inhoud van register H.

De operatiecode is slechts één byte en wordt gegeven als

76 543 210 bitnummer.
01 r.. s.. byte 1.

Voor zowel r als s moeten drie bits worden ingevuld die te vinden zijn in de volgende tabel:

r,s	reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

Voor de instructie

LD B,H

moet de operatiecode zijn:

76 543 210 (bitnummer)
01 000 100 (44H)

Een tweebyte instructie is

LD r,n

Laad het processorregister r met de waarde n.

Hier wordt gebruikgemaakt van de immediate addressing mode (paragraaf 4.2.) waarbij de operand direct na de operatiecode in het programmageheugen is geplaatst. Deze instructie wordt gebruikt indien een achtbits getal moet worden ingevoerd dat tijdens het verloop van het programma niet behoeft te veranderen. De operatiecode kan worden gevonden met

76 543 210 bitnummer.
00 r.. 110 byte1.

Ook nu moeten voor r drie bits worden ingevuld. Het zijn dezelfde bits die in de hiervoor gegeven tabel zijn benoemd.

De register indirect addressing mode (paragraaf 4.6) is toegepast bij de instructie

LD r,(HL)

Laad het processorregister r met de inhoud van de geheugenplaats, waarvan het adres wordt gegeven door de inhoud van het registerpaar HL.

Dit is weer een éénbyte instructie waarvan de operatiecode te vormen is met

76 543 210 bitnummer.
01 r.. 110 byte1.

en de hiervoor gegeven tabel.
Het adres dat door de inhoud van het registerpaar HL wordt gegeven kan een register zijn van het datageheugen. Het hierin opgeslagen getal (data) kan eventueel tijdens het verloop van het programma worden veranderd.
Soms heeft een instructie een "tegenvoeter", namelijk een instructie waarbij het doel- en het bronregister zijn verwisseld:

LD (HL),r

Laad het register, waarvan het adres is gegeven door de inhoud van het registerpaar HL, met de inhoud van het processorregister r.

Ook nu wordt de register indirect addressing mode toegepast. De instructie is een éénbyte instructie waarvan de operatiecode kan worden gevonden met

76 543 210 bitnummer.
01 110 r.. byte1.

De drie bits die u voor r moet invullen kunt u weer vinden met behulp van de voorgaande tabel.

Voor de instructie

LD (HL),D

zal de volgende operatiecode moeten worden gebruikt:

76 543 210 bitnummer
01 110 010 (72H)

Instructies die met de voorgaande geheel overeenkomen zijn

LD A,(BC)

en

LD A,(DE)

Ook de registerparen BC en DE zijn dus voor indirecte adressering te gebruiken, in dit geval echter uitsluitend met als doelregister de Accu. Bij deze instructies behoeven de operatiecodes niet zelf te worden samengesteld maar kunnen direct uit de instructieset worden afgelezen. Hierin vinden we voor

LD A,(BC)

de operatiecode 00001010 (0AH)
en voor

LD A,(DE)

de operatiecode 00011010 (1AH).

De tegenvoeters van deze instructies zijn

LD (BC),A

met de operatiecode 00000010 (02H)
en

LD (DE),A

met de operatiecode 00010010 (12H).

De register indirect/immediate addressing mode is toegepast bij de instructie

LD (HL),n

Laad het register, waarvan het adres is gegeven door de inhoud van het registerpaar HL, met de waarde n.

De operatiecode hiervoor is 00110110 (36H).

Het getal dat in het doelregister wordt geladen volgt weer direct op de operatiecode in het programmageheugen en moet daarom een onveranderlijk getal zijn.

De transportinstructies die werken met de register indirect addressing mode hebben geen invloed op de waarde van de FLAGS.

De adresseermethode die bij de volgende instructies is toegepast lijkt sprekend op de register addressing mode. In de instructieset wordt voor deze instructies echter implied als de addressing mode aangegeven.

LD A,I

Laad de Accu met de inhoud van het interruptregister.

Operatiecode 11101101 01010111 (ED57H).

LD I,A

Laad het interruptregister met de inhoud van de Accu.

Operatiecode 11101101 01000111 (ED47H).

Deze twee instructies zijn de enige transportinstructies die invloed hebben op de FLAGS. Hoe deze invloed is kan worden afgelezen uit de instructieset.

Bij bovenstaande tweebytes instructies is behalve de Accu ook het interruptregister I betrokken. Dit register speelt een rol bij de maskeerbare interrupt mode 2 (paragraaf 5.6) en kan met LD I,A van de juiste inhoud worden voorzien. Het interruptregister kan echter ook voor tijdelijke opslag van data worden gebruikt en daarom is LD A,I toegevoegd om deze data weer in de Accu terug te kunnen schrijven. We zouden dit laatste als een ''oneigenlijk gebruik'' van een register kunnen beschouwen. Ook het Refreshregister R kan worden gebruikt voor tijdelijke data- opslag. Het ''eigenlijke gebruik'' van dit register zult u bij uw MSX-computer waarschijnlijk wel niet tegenkomen. Uw computer is voorzien van *''statische''* RAM geheugenelementen die slechts hun inhoud verliezen als de computer wordt uitgeschakeld. Er zijn ook zogenaamde *''dynamische''* RAM geheugenelementen. Deze verliezen, ook bij ingeschakelde voeding, na enige tijd hun inhoud en moeten daarom voortdurend opnieuw worden ingeschreven, *''refreshd''*.

Hierbij wordt het register R in de processor gebruikt. Voor data-uitwisseling tussen de Accu en het register R kunnen de volgende instructies worden gebruikt in de implied addressing mode:

LD A,R

Laad de Accu met de inhoud van het refreshregister.

Operatiecode 11101101 01011111 (ED5FH).

LD R,A

Laad het refreshregister met de inhoud van de Accu.

Operatiecode 11101101 01001111 (ED4FH).

Ook dit zijn tweebyte instructies maar deze hebben in tegenstelling tot de voorgaande geen invloed op de waarde van de FLAGS.

De absolute addressing mode wordt in de instructieset de extended addressing mode genoemd. Dit heeft betrekking op het zestienbits adres dat direct na de operatiecode moet volgen (paragraaf 4.4). De volgende instructie gebruikt deze mode:

LD A,(nn)

Laad de Accu met de inhoud van het register dat door de bytes nn is geadresseerd.

Operatiecode 00111010 (3AH).

Dit is een driebytes instructie omdat direct na de operatiecode nog twee bytes moeten volgen die het adres van het bronregister aangeven. Dit adres staat in de volgorde lage byte - hoge byte in het geheugen opgeslagen. Deze instructie heeft geen invloed op de waarde van de FLAGS. Dat is ook niet het geval met zijn tegenvoeter

LD (nn),A

Laad het register dat door de bytes nn is geadresseerd met de inhoud van de Accu.

Operatiecode 00110010 (32H).

Bij deze instructie moet de operatiecode in het programmageheugen direct worden gevolgd door twee bytes, die in de volgorde lage byte - hoge byte het adres van het doelregister geven. Er zijn daarom voor de gehele instructie drie bytes nodig.

Twee instructies die gebruikmaken van de indexed addressing mode (paragraaf 4.11) zijn

LD r,(IX + d)
LD r,(IY + d)

Laad het processorregister r met de inhoud van de geheugenplaats waarvan het adres wordt gevonden door de inhoud van het indexregister te vermeerderen met de waarde d.

De operatiecodes voor de instructies moeten weer worden samengesteld. De operatiecode voor LD r,(IX + d) met

76	543	210	bitnummer.
11	011	101	byte1.
01	r..	110	byte2.
d.	...	...	byte3.

De operatiecode voor LD r,(IY + d) is te vinden met

76	543	210	bitnummer.
11	111	101	byte1.
01	r..	110	byte2.
d.	...	...	byte3.

Voor r moeten drie bits worden ingevuld die afhankelijk zijn van het gekozen register. De volgende tabel kan hiervoor worden gebruikt:

r..	reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

De instructies zijn driebytes instructies omdat direct na de twee bytes voor de operatiecode nog een byte moet volgen dat de waarde van d inhoudt. De waarde van d is een constante zodat voor het adresseren van diverse geheugenplaatsen steeds de inhoud van het indexregister zal moeten worden veranderd. De instructies hebben geen invloed op de waarde van de FLAGS.

De tegenvoeters van de voorgaande instructies zijn:

LD (IX+d),r
LD (IY+d),r

Laad het register waarvan het adres wordt gevonden door de inhoud van het indexregister te vermeerderen met de waarde van d, met de inhoud van het processorregister r.

Voor het samenstellen van de operatiecode kan de bovenstaande tabel voor r worden gebruikt.

LD (IX+d),r:

76	543	210	bitnummer.
11	011	101	byte1.
01	110	r..	byte2.
d.	...	...	byte3.

LD (IY+d),r:

76	543	210	bitnummer.
11	011	101	byte1.
01	110	r..	byte2.
d.	...	...	byte3.

De laatste instructies van deze groep zijn

LD (IX+d),n
LD (IY+d),n

Laad het register waarvan het adres wordt gevonden door de inhoud van het indexregister te vermeerderen met de waarde d, met de waarde n.

De hier toegepaste adresseringsmethode is indexed/immediate.
De instructies hebben geen invloed op de FLAGS.

De operatiecodes zijn:

LD (IX+d),n:

76	543	210	bitnummer.
11	011	101	byte1.
00	110	110	byte2.
d.	...	...	byte3.
n.	...	...	byte4.

LD (IY+d),n:

76	543	210	bitnummer.
11	111	101	byte1.
00	110	110	byte2.
d.	...	...	byte3.
n.	...	...	byte4.

SOURCE

DESTINATION		IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR.	IMME.
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	6E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDEXED	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT. ADDR	(nn)			32 n n													
IMPLIED	I			ED 47													
	R			ED 4F													

Lijst 3.
Achtbits LOAD-instructies.

b. zestienbits LOAD-instructies.
De zestienbits LOAD-instructies hebben steeds betrekking op een *registerpaar* (BC, DE, HL) of een van de zestienbitsregisters IX, IY en de Stackpointer SP uit de processor. Omdat de databus achtbits is zullen steeds twee achtbits getallen, een hoge byte en een lage byte, naar of van het desbetreffende register(paar) moeten worden getransporteerd. Normaal gesproken zouden daarvoor twee instructies nodig zijn, voor elke byte een. De zestienbits LOAD-instructies van de Z80 processor maken het echter mogelijk de genoemde registers met één instructie van hun inhoud te voorzien.

De data die in een registerpaar aanwezig is kent steeds een lage en een hoge byte. Bij het registerpaar BC vinden we de hoge byte in register B en de lage byte in register C. Op overeenkomstige wijze is de data in de registerparen DE en HL georganiseerd.

Er zijn drie instructies die werken met de register extended addressing mode (paragraaf 4.7).

LD SP,HL

Opcode: 11111001 (F9H).

LD SP,IX

Opcode: 11011101 11111001 (DDF9H).

LD SP,IY

Opcode: 11111101 11111001 (FDF9H).

Bij deze instructies zijn alleen processorregisters betrokken en zij hebben steeds het doel om de Stackpointer van een bepaalde inhoud te voorzien. Op deze wijze kan de plaats van de STACK in de geheugenruimte worden bepaald.

De immediate extended addressing mode (paragraaf 4.3) is toegepast bij de instructie

LD dd,nn

Hier wordt het registerpaar dd (BC, DE, HL en SP) geladen met de twee bytes die in aansluiting op de operatiecode in het programmageheugen zijn geladen. Op het eerstvolgende adres na de operatiecode vinden we de lage byte en op het daaropvolgende adres de hoge byte.
De operatiecode kan worden samengesteld met

76	543	210	bitnummer.
00	dd0	001	byte1.
n.	...	...	byte2.
n.	...	...	byte3.

Voor dd moeten twee bits worden ingevuld, die te vinden zijn in de volgende tabel:

dd	paar
00	BC
01	DE
10	HL
11	SP

Deze tabel wordt ook nog bij andere adresseringsmethoden gebruikt. U ziet dat de Stackpointer ook nog op een andere wijze van zijn inhoud kan worden voorzien dan alleen maar door middel van de register extended addressing mode.

Behalve de reeds genoemde registers kunnen ook de indexregisters, met gebruikmaking van de immediate extended addressing mode, worden geladen met data.

LD IX,nn

Operatiecode:

76	543	210	bitnummer.
11	011	101	(DDH) byte1.
00	100	001	(21H) byte2.
n.	...	...	byte3.
n.	...	...	byte4.

LD IY,nn

Operatiecode:

76	543	210	bitnummer.
11	111	101	(FDH) byte1.
00	100	001	(21H) byte2.
n.	...	...	byte3.
n.	...	...	byte4.

Bij de absolute extended addressing mode (paragraaf 4.5.) volgen direct na de operatiecode de lage en de hoge byte van een adres in het datageheugen. Op dit adres vinden we de lage byte van de data die in het desbetreffende register moet worden geladen. De hoge byte van de data volgt direct hierna.

LD dd,(nn)

Hierin is dd één van de registerparen BC, DE, HL of SP. De operatiecode kan worden samengesteld met

76	543	210	bitnummer.
11	101	101	byte1.
01	dd1	011	byte2.
n.	...	...	byte3.
n.	...	...	byte4.

Voor dd kunnen de twee bits worden ingevuld die gevonden kunnen worden in de hiervoor genoemde tabel. Hoewel ook het registerpaar HL hieronder valt is er voor dit registerpaar nog een afzonderlijke instructie:

LD HL,(nn)

Operatiecode:

76	543	210	bitnummer.
00	101	010	byte1.
n.	...	...	byte2.
n.	...	...	byte3.

U ziet dat deze instructie het voordeel heeft een byte minder nodig te hebben dan de vorige.

Ook de indexregisters kunnen op deze manier van hun inhoud worden voorzien.

LD IX,(nn)

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
00	101	010	byte2.
n.	...	...	byte3.
n.	...	...	byte4.

LD IY,(nn)

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
00	101	010	byte2.
n.	...	...	byte3.
n.	...	...	byte4.

Alle transportinstructies die gebruikmaken van de absolute extended addressing mode kennen een tegenvoeter:

LD (nn),dd

Operatiecode:

76	543	210	bitnummer.
11	101	101	byte1.
01	dd0	011	byte2.
n.	...	...	byte3.
n.	...	...	byte4.

In dit geval bevatten de twee bytes die in het programmageheugen direct volgen op de operatiecode, het adres van het doelregister waarin de inhoud van het desbetreffende processorregisterpaar moet worden opgeslagen. Voor dd moeten in byte2 van de operatiecode weer de twee bits worden ingevuld die in de eerder genoemde tabel te vinden zijn.

LD (nn),HL

Operatiecode:

76	543	210	bitnummer.
00	100	010	byte1.
n.	...	...	byte2.
n.	...	...	byte3.

LD (nn),IX

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
00	100	010	byte2.
n.	...	...	byte3.
n.	...	...	byte4.

LD (nn),IY

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
00	100	010	byte2.
n.	...	...	byte3.
n.	...	...	byte4.

De zestienbits LOAD-instructies zijn samengevat in lijst 4.

c. De STACK-instructies.
Zoals in hoofdstuk 5 is uiteengezet wordt door een interrupt een programma op een niet van tevoren te bepalen plaats onderbroken. De processorregisters kunnen dan gegevens bevatten die voor het

		SOURCE									
		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
DESTINATION REGISTER	AF										F1
	BC								01 n n	ED 4B n n	C1
	DE								11 n n	ED 5B n n	D1
	HL								21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
PUSH INSTRUCTIONS → REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5			

↑ POP INSTRUCTIONS

Lijst 4. Zestienbits LOAD-instructies en STACK-instructies.

verdere verloop van het programma noodzakelijk zijn en die verloren zouden gaan als het interruptprogramma zonder meer deze registers zou gaan gebruiken, zonder eerst de inhoud daarvan ergens in het geheugen op te slaan. Voor het tijdelijk opslaan van deze gegevens (gedurende het afwerken van het interruptprogramma) is het STACK bijzonder geschikt. Direct bij de aanvang van het interruptprogramma wordt dan de inhoud van de processorregisters die ook door het interruptprogramma worden gebruikt, naar de STACK geschreven. Hiervoor zijn de éénbyte PUSH-instructies. Geheel aan het eind van het interruptprogramma moeten de instructies gegeven worden om de processorregisters weer de oorspronkelijke inhoud te geven en waarmee de inhoud van de desbetreffende STACK-registers weer naar de processorregisters kan worden getransporteerd. Dit zijn de éénbyte POP-instructies. De STACK-transportinstructies hebben geen invloed op de waarde van de FLAGS.

Elke instructie heeft betrekking op een registerpaar BC, DE, HL of AF. In dit geval zijn de Accu en het FLAG register F ook tot een registerpaar gevormd.
Ook de inhoud van de zestienbits registers IX en IY kunnen door middel van een enkele instructie naar de STACK worden geschreven of uit de STACK weer van de juiste inhoud worden voorzien. Gebruik wordt gemaakt van de register indirect addressing mode.

PUSH qq

Operatiecode:

76	543	210	bitnummer.
11	qq0	101	byte1.

In de operatiecode dienen de twee bits qq te worden ingevuld afhankelijk van het desbetreffende registerpaar. De bits kunnen worden gevonden in de volgende tabel:

qq	paar.
00	BC
01	DE
10	HL
11	AF

Het adres van de STACK waar de inhoud van het desbetreffende registerpaar moet worden bewaard wordt aangegeven door de inhoud van de Stackpointer SP. Stel dat de inhoud van SP gelijk is aan 97FE en dat het registerpaar BC naar de STACK moet worden geschreven. Het adres dat de Stackpointer aangeeft is het laatste dat is ingeschreven en de eerste vrije geheugenplaats in de STACK is 97FD. Als eerste handeling verlaagt de processor de inhoud van SP. De inhoud van SP wordt daarna in de adresbuffers geladen waarna de inhoud van het register B via de databus in de STACK op het adres 97FD wordt geschreven. De inhoud van SP wordt opnieuw met 1 verlaagd en in de adresbuffers geplaatst. Deze geven het adres 97FC aan en op dit adres wordt de inhoud van het register C geladen. Fig. 47 geeft de inhoud van de desbetreffende registers weer.
De andere registerparen worden door de desbetreffende instructies op overeenkomstige wijze behandeld.

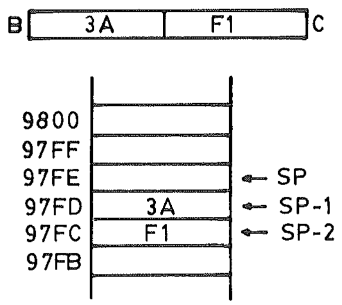


Fig. 47. De werking van de STACK.

Ook de inhoud van de indexregisters kan in de STACK worden bewaard:

PUSH IX

Opcode: 11011101 11100101 (DDE5H).

PUSH IY

Opcode: 11111101 11100101 (FDE5H).

Uiteraard is de omgekeerde richting ook mogelijk:

POP qq

Operatiecode:

76	543	210	bitnummer.
11	qq0	001	byte1.

De bits voor qq vindt u weer in de hiervoor gegeven tabel. Stellen we dat het registerpaar BC weer van de oorspronkelijke inhoud moet worden voorzien, dan zal SP het adres 97FC moeten aangeven. De inhoud van dit adres wordt in register C geladen. Nu wordt de inhoud van SP met 1 opgehoogd zodat het adres 97FD wordt aangewezen. De inhoud hiervan gaat naar register B. Als laatste wordt opnieuw de Stackpointer met 1 verhoogd. Deze geeft nu het adres 97FE aan. Dat is weer de oorspronkelijke situatie van voor de PUSH-instructie.

Voor de indexregisters hebben we de instructies

POP IX

Opcode: 11011101 11100001 (DDE1H).

POP IY

Opcode: 11111101 11100001 (FDE1H).

De STACK-instructies zijn vooral belangrijk bij het gebruiken van interruptroutines en subroutines. Worden bij het uitvoeren van een interruptroutine bepaalde processorregisters gebruikt, dan zal de data die hierin door het hoofdprogramma was opgeslagen verloren gaan. Dit geldt in elk geval voor de Accu en het FLAG- register. Daarom moeten we direct aan het begin van de interruptroutine de inhoud van de processorregisters die worden gebruikt, naar de STACK schrijven. Stel dat de registers B, C, D, E, H en L in de interruptroutine worden gebruikt, dan luiden de eerste instructies:

PUSH AF
PUSH BC
PUSH DE
PUSH HL

Aan het eind van de interruptroutine moeten de registers hun oorspronkelijke inhoud terug krijgen om het hoofdprogramma weer te kunnen gaan vervolgen. Hiervoor zijn de instructies:

POP HL
POP DE
POP BC
POP AF

Merk op dat de volgorde van de POP-instructies tegengesteld is aan die van de PUSH-instructies. In

overeenstemming met de werking van de STACK: last in - first out is dat nodig om de juiste data ook weer in het juiste register terug te schrijven.

De STACK-instructies zijn samengevat in lijst 4.

d. De dataverwisselinstructies (EXCHANGE).
Bij de tot nu toe behandelde datatransportinstructies is het resultaat van elke instructie dat er uiteindelijk twee registers zijn die beide dezelfde inhoud hebben. De inhoud van het doelregister wordt hetzelfde als die van het bronregister terwijl de inhoud van het bronregister gelijk blijft. Bij de dataverwisselinstructies is er eigenlijk geen sprake van een bron- en een doelregister. De werking van deze groep instructies volgt direct uit de naam: de inhoud van twee registers wordt onderling verwisseld. Stel dat we de data in de registerparen DE en HL moeten verwisselen. De volgende instructies zouden hiervoor gebruikt kunnen worden:

PUSH DE
PUSH HL
POP DE
POP HL

Merk op dat de volgorde van de POP-instructies nu niet tegengesteld is aan die van de PUSH-instructies!

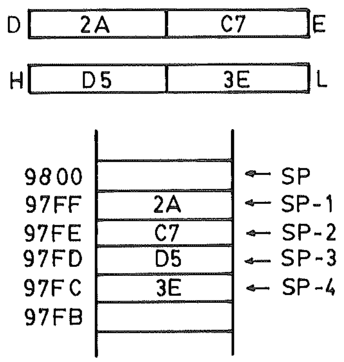


Fig. 48. Verwisselen van registerinhouden, eerste fase.

Stel dat bij het begin van dit programmaatje de Stackpointer het adres 9800H aangeeft. Door de instructie PUSH DE wordt de inhoud van SP eerst verlaagd tot 97FF. Naar dit adres wordt de inhoud van register D geschreven. Daarna wordt de inhoud van register E in het adres 97FE geladen. PUSH HL heeft een overeenkomstige werking zodat hierna de situatie is zoals geschetst in fig. 48. Bij het begin van de POP DE instructie heeft SP als inhoud 97FC. De data op dit adres wordt in register E geladen (lage byte). De Stackpointer wordt met 1 verhoogd tot 97FD, waarna register D zijn nieuwe inhoud krijgt. Als laatste werking van de instructie wordt SP weer met 1 verhoogd tot 97FE. De werking van POP HL komt hiermee overeen, zodat de nieuwe inhoud van het registerpaar HL gelijk is aan die van registerpaar DE voor de aanvang van dit programma. SP geeft uiteindelijk weer het adres 9800 aan (fig. 49).

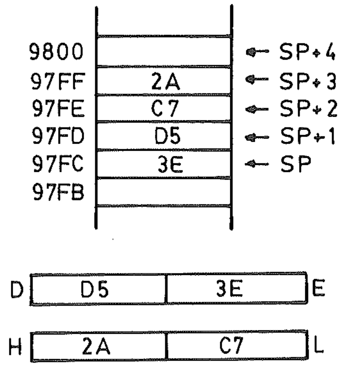


Fig. 49. Verwisselen van geheugeninhouden, tweede fase.

Deze vier instructies kunnen worden vervangen door één enkele:

EX DE,HL

Operatiecode: 11101011 (EB).

Bovenstaande instructie werkt met de implied addressing mode. De volgende instructies maken gebruik van de register indirect addressing mode voor het verwisselen van de inhoud van een registerpaar met die van bepaalde STACK-registers.

EX (SP),HL

Operatiecode: 11100011 (E3H).

De werking van de instructie is dat eerst de inhoud van het Stackadres dat door SP wordt aangegeven, wordt verwisseld met de inhoud van het register L, waarna de inhouden van de registers SP + 1 en H worden verwisseld. Nemen we aan dat de inhoud van SP gelijk is aan 97FC, dan geeft fig. 50 de si-

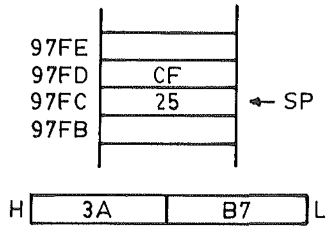


Fig. 50. Voor de verwisseling.

tuatie voor, en fig. 51 de situatie na de verwisseling aan.
Door de instructie wordt de inhoud van de Stackpointer *niet* veranderd! Bij het voorbeeld van de figuren 50 en 51 *blijft* de inhoud van de Stackpointer dus 97FC.

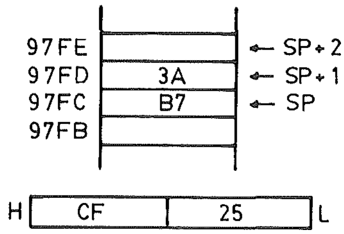


Fig. 51. Na de verwisseling.

Overeenkomstig hiermee werken de instructies

EX (SP),IX

Operatiecode 11011101 11100011 (DDE3H).

EX (SP),IY

Operatiecode 11111101 11100011 (FDE3H).

		IMPLIED ADDRESSING				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

Lijst 5. Dataverwisselinstructies.

In paragraaf 3.6. is de aanwezigheid van de alternatieve processorregisters A', B', C', D', E', H', L' en F' vermeld. Het enige waarvoor deze registers kunnen worden gebruikt is voor het uitwisselen van hun inhouden met de overeenkomstige processorregisters. Dit verwisselen geschiedt volgens de implied addressing mode.

EX AF,A'F'

Operatiecode: 00001000 (08H).

Deze instructie heeft tot gevolg dat de inhoud van Accu A met die van het register A' en de inhoud van het FLAG-register F met die van register F' wordt verwisseld.

EXX

Operatiecode: 11011001 (D9H).

Deze instructie veroorzaakt het verwisselen van de inhoud van de processorregisters B, C, D, E, H en L met die van de overeenkomstige alternatieve registers (B met B', C met C' enz.). Lijst 5 geeft een samenvatting van de dataverwisselinstructies.

6.3. Rekenkundige instructies.

De *arithmetische* of rekeninstructies zijn in twee groepen onder te verdelen:

a. Het optellen of aftrekken van twee getallen.
b. Het verhogen of verlagen van een registerinhoud met 1.

In principe kan de processor alleen maar optellen. Het aftrekken komt tot stand door het optellen van een negatief getal (paragraaf 2.2.). Toch is dat voldoende om de computer de meest ingewikkelde berekeningen te laten maken. Een vermenigvuldiging komt tot stand door herhaald optellen, zodat door het invoeren van het juiste programma praktisch elke berekening mogelijk wordt.

Bij het optellen van twee getallen speelt het Carrybit een grote rol. Het Carrybit wordt gebruikt als het resultaat van een optelling te groot is voor een register. De waarde van het Carrybit moet dan ook worden gebruikt bij het optellen van twee getallen die groter zijn dan acht bits. Voor het optellen van twee getallen is er de instructie

ADD.

Eén van de twee getallen moet aanwezig zijn in de Accu. Eventueel kan dit met een transportinstructie in de Accu zijn gebracht. Het tweede getal bevindt zich in het programmageheugen of in het data geheugen en wordt, afhankelijk van de adresseermethode, in het hulpregister TMP geladen. Daarna vindt de optelling plaats waarbij het resultaat van de optelling uiteindelijk weer in de Accu staat (paragraaf 3.5.). De meeste van de rekeninstructies hebben invloed op de waarde van de FLAGS in het Flagregister. Dit is nodig omdat een geconditioneerde sprong vaak afhankelijk is van de uitkomst van een berekening, terwijl de processor het eventueel uitvoeren van de sprong afhankelijk laat zijn van de FLAGS.

In het volgende voorbeeld zijn twee positieve getallen van acht bits bij elkaar opgeteld:

00011011	getal a.
01110111	getal b.
10010010	som s.

Het resultaat van de FLAGS is:

C=0. Het resultaat is niet groter dan achtbits.
Z=0. Het resultaat van de berekening is niet nul.
S=1. Het achtste bit (bit 7) van het resultaat is 1.
V=1. Het optellen van twee positieve getallen kan nooit een negatief resultaat opleveren. De FLAG S is echter 1, hetgeen zou duiden op een negatief resultaat. Omdat deze aanduiding onjuist is wordt FLAG V "1" gemaakt.
Ook de FLAGS H en N worden beïnvloed. Die hebben in dit geval echter geen betekenis.

De volgende instructies zijn voor deze optelling nodig:

LD A,adres getal a.
ADD,adres getal b.

Omdat het resultaat zich in de Accu bevindt, kan eventueel nog een transportinstructie nodig zijn om het resultaat naar het datageheugen te schrijven:

LD adres,A

Het optellen van twee getallen van zestien bits met de Accu moet in twee stappen gebeuren daar de Accu, evenals het hulpregister, slechts getallen van achtbits kan bevatten. In het volgende voorbeeld worden twee positieve getallen opgeteld:

00111100	10100010	getal a.
01010101	10111100	getal b.
10010010	01011110	som s.

C=0, Z=0, S=1, V=1

Beide getallen zijn in een hoge en een lage byte te splitsen. De waarde van de FLAGS S en V is hier bepaald door bit 7 van de hoge byte van het resultaat.
De optelling dient door de processor als volgt te worden uitgevoerd:

1^{e} stap, lage bytes.

10100010	lage byte getal a.
10111100	lage byte getal b.
01011110	lage byte som.

C=1, Z=0, S=0, V=1.

De processor denkt te doen te hebben met twee negatieve getallen en een positief resultaat (S=0). Daarom wordt V "1".
Bij de tweede stap van de optelling moet rekening worden gehouden met de waarde die de Carry bij de eerste optelling heeft gekregen. De Carry dient bij de hoge bytes te worden opgeteld.

2^{e} stap, hoge bytes.

00000001	Carrybit.
00111100	hoge byte getal a.
01010101	hoge byte getal b.
10010010	hoge byte som.

C=0, Z=0, S=1, V=1.

Resultaat: 10010010 01011110

De inhoud van C, S en V is bepalend voor de *gehele* som. Dat geldt niet voor Z, zoals blijkt uit het volgende voorbeeld waarin een negatief getal bij een positief getal is opgeteld.

00111100	10100010	getal a (positief).
11000011	10111100	getal b (negatief).
00000000	01011110	som s.

C = 1, Z = 0, S = 0, V = 0.

Het resultaat (de gehele som) is niet nul, zodat besloten is voor Z = 0 (denk er om: als het resultaat 0 is wordt de FLAG Z geset, Z = 1). Het resultaat is geheel correct, S = 0 en dus V = 0. Met de waarde van C mag hier geen rekening worden gehouden. De processor voert de berekening als volgt uit:

1e stap lage bytes.

10100010	lage byte getal a.
10111100	lage byte getal b.
01011110	lage byte som s.

C = 1, Z = 0, S = 0, V = 1.

2e stap, hoge bytes.

00000001	Carrybit.
00111100	hoge byte getal a.
11000011	hoge byte getal b.
00000000	hoge byte som s.

C = 1, Z = 1, S = 0, V = 0.

Resultaat: 00000000 01011110

Het resultaat van de tweede stap van de berekening is 0, zodat door de processor de Z FLAG wordt geset. Dit is niet correct voor wat betreft de gehele som.

Als laatste voorbeeld een optelling van twee negatieve getallen.

11101101	11101011	getal a.
11101110	11001110	getal b.
11011100	10111001	som s.

C = 1, Z = 0, S = 1, V = 0.

1e stap, lage bytes.

11101011	lage byte getal a.
11001110	lage byte getal b.
10111001	lage byte som s.

C = 1, Z = 0, S = 1, V = 0.

2e stap, hoge bytes.

00000001	Carrybit.
11101101	hoge byte getal a.
11101110	hoge byte getal b.
11011100	hoge byte som s.

C = 1, Z = 0, S = 1, V = 0.

Resultaat: 11011100 10111001

De instructie die voor de eerste stap van de optelling beschikbaar is, is al bekend, ADD. Voor de tweede stap is een andere instructie nodig omdat behalve de twee getallen ook nog de Carry moet worden opgeteld. De instructie die hiervoor kan worden gebruikt luidt:

ADC

De instructievolgorde die voor het optellen van twee zestienbits getallen moet worden gebruikt is:

```
LD A,adres aL
ADD A,adres bL
LD adres sL,A
LD A,adres aH
ADC A,adres bH
LD adres sH,A
```

Het is gebruikelijk dat de hoge byte en de lage byte van een getal in twee elkaar opeenvolgende geheugenplaatsen zijn geladen. Zo zal uiteindelijk ook het resultaat van de berekening gevonden worden in twee elkaar opeenvolgende geheugenplaatsen (adres sL, adres sH).

De bewerking vindt steeds plaats tussen een getal in de Accu en een getal in een register van de processor, het datageheugen of van het programmageheugen dat, afhankelijk van de gebruikte adresseermethode, in het hulpregister TMP wordt geladen.

ADD A,r (register addressing mode).

Operatiecode:

76	543	210	bitnummer.
10	*000*	r..	byte1.

Voor r dienen drie bits te worden ingevoerd die gevonden kunnen worden in onderstaande tabel.

r..	reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

ADD A,n (Immediate)

Operatiecode:

76	543	210	bitnummer.
11	*000*	110	byte1.
n.	...	...	byte2.

ADD A,(HL) (register indirect)

Operatiecode:

76	543	210	bitnummer.
10	*000*	110	(86H).

ADD A,(IX + d) (indexed)

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
10	*000*	110	byte2.
d.	...	...	byte3.

ADD A,(IY + d) (indexed)

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
10	*000*	110	byte2.
d.	...	...	byte3.

ADC A,s

Bij deze instructie zijn dezelfde adresseermethoden mogelijk als bij ADD. Daarom kan voor s gelezen worden: r, n, (HL), (IX + d) of (IY + d). De operatiecoden zijn op dezelfde wijze samen te stellen als die voor ADD. Nu moeten de bits 5, 4 en 3 (let op de volgorde) vervangen worden door 001. Dit zijn de cursief gedrukte bits. Bij de indexed addressing mode geldt dit alleen voor de overeenkomstige bits van byte2.

Het aftrekken van twee getallen wordt door de processor uitgevoerd in het twee-complement systeem, zoals verklaard in paragraaf 2.2. De manier waarop de negatieve waarde van een binair getal kan worden gevonden is in deze paragraaf ook beschreven: alle bits van het getal inverteren en er daarna 1 bij optellen. Deze methode wordt dan ook gevolgd bij de volgende voorbeelden.

01101001	getal a.
01001100	getal b.
00011101	verschil.

Dit zijn twee positieve getallen en het resultaat is eveneens positief.
De processor verwerkt deze opgave als volgt:

00000001	Carrybit.
01101001	getal a.
10110011	NOT(b).
00011101	verschil v.

C = 1, Z = 0, S = 0, V = 0.

Eerst wordt het Carrybit "1". Dan worden alle bits van getal b geïnverteerd (de 0 wordt 1 en de 1 wordt 0). Het resultaat hiervan wordt met NOT(b) aangegeven. Daarna worden het Carrybit, getal a en NOT(b) bij elkaar opgeteld. Het resultaat is het verschil v. Dat na de bewerking het Carrybit 1 is heeft hier verder geen betekenis.

Het volgende voorbeeld geeft twee positieve getallen met een negatief resultaat.

getal a: 01001100, getal b: 01101001.

00000001	Carrybit.
01001100	getal a.
10010110	NOT(b).
11100011	verschil v.

C = 0, Z = 0, S = 1, V = 0.

Een positief getal (b) aftrekken van een negatief (a).

Getal a: 10110110, getal b: 01011001.

00000001 Carrybit.
10110110 getal a.
10100110 NOT(b).

01011101 Verschil v.

C=1, Z=0, S=0, V=1.

In het resultaat is bit 7 nul, dus S=0. Het is een aftrekking waarvan het resultaat negatief is (het optellen van twee negatieve getallen), dus V=1.

Een negatief getal aftrekken van een positief.

Getal a: 01011001, getal b: 10110110

00000001 Carrybit.
01011001 getal a.
01001001 NOT(b).

10100011 verschil v.

C=0, Z=0, S=1, V=1.

De bewerking resulteert in het optellen van twee positieve getallen. Omdat bit 7 van het resultaat 1 is wordt FLAG S "1", vandaar V=1. Dat geeft aan dat het resultaat een positief getal is.

Een negatief getal aftrekken van een negatief getal met een positief resultaat.

Getal a: 11010100, getal b: 10010010.

00000001 Carrybit.
11010100 getal a.
01101101 NOT(b).

01000010 verschil v.

C=1, Z=0, S=0, V=0.

De bovenstaande aftrekkingen worden volledig uitgevoerd (inclusief het "1" maken van het Carrybit) door de instructie

SUB s

Bij deze bewerking bevindt het aftrektal (getal a) zich in de Accu. De aftrekker (getal b) is de operand die door s wordt geadresseerd.
De volgorde van de instructies wordt:

LD A,adres a
SUB adres b
LD adres v,A

Bij deze instructie kunnen dezelfde adresseermethoden worden toegepast als bij ADD A,s en in SUB s kan voor s daarom worden gelezen r, n, (HL), (IX+d) of (IY+d). Als de drie cursief gedrukte bits in de diverse operatiecoden voor ADD A,s worden vervangen door 010, dan vindt u de operatiecoden voor SUB s.

Het aftrekken van getallen die groter zijn dan acht bits moet weer in een aantal stappen gebeuren. In het volgende voorbeeld gaat het om twee zestienbits getallen. Uiteraard moet de bewerking in twee stappen geschieden.

Getal a: 01101010 00011101
Getal b: 01001011 01101011

1e stap:

00000001 Carrybit.
00011101 getal aL.
10010100 NOT(bL).

10110010 verschil vL.

C=0, Z=0, S=1, V=0.

2e stap:

00000000 Carrybit.
01101010 getal aH.
10110100 NOT(bH).

00011110 verschil vH.

C=1, Z=0, S=0, V=0.

Verschil v: 00011110 10110010

De waarde van de Carry bij de tweede stap van de aftrekking is afhankelijk van het resultaat van de eerste stap. Voor de tweede stap is daarom een andere instructie nodig dan voor de eerste:

SBC A,s

Ook nu kan voor s r, n, (HL), (IX+d), of (IY+d) worden ingevuld. De operatiecode kan op overeenkomstige wijze als die voor ADD A,s worden gevonden. In dit geval moeten de cursief gedrukte bits echter worden vervangen door 011.
De instructievolgorde voor het vorige voorbeeld zal moeten zijn:

LD A,adres aL
SUB adres bL
LD adres vL,A
LD A,adres aH
SBC A,adres bH
LD adres vH,A

Een instructie die veel wordt gebruikt bij het maken van een teller is INC. Door deze instructie wordt het geadresseerde register met 1 verhoogd (bij de inhoud wordt 1 opgeteld). De instructie kan bijvoorbeeld worden gebruikt indien moet worden bijgehouden hoeveel keren een bepaalde handeling door de computer is uitgevoerd. Er zijn drie adresseermethoden bij mogelijk.

INC r

Operatiecode:

76	543	210	bitnummer.
00	r..	*100*	byte1.

De drie bits die voor r moeten worden ingevoerd zijn weer te vinden in de tabel die bij ADD gegeven is.

INC (HL)

Operatiecode: 00 110 *100*

INC (IX + d)

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
00	110	*100*	byte2.
d.	...	...	byte3.

INC (IY + d)

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
00	110	*100*	byte2.
d.	...	...	byte3.

	SOURCE										
	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

Lijst 6. De achtbits reken- en logische instructies.

Aftellen is ook mogelijk, dat wil zeggen dat in dat geval de inhoud van het geadresseerde register met 1 wordt verminderd.

DEC r

Hierbij zijn dezelfde adresseermethoden toe te passen als bij INC. De operatiecoden zijn te vinden door de cursief gedrukte bits in de overeenkomstige operatiecode voor INC te vervangen door 101.

De achtbitsrekeninstructies zijn samengevat in lijst 6.

Bovenstaande rekeninstructies hadden steeds betrekking op achtbits registers. Er zijn ook rekeninstructies die betrekking hebben op zestienbits register(paren) in de processor.

ADD HL,ss

In dit geval heeft het registerpaar HL schijnbaar dezelfde werking als de Accu. De bewerking vindt plaats tussen HL en het geadresseerde register terwijl na de bewerking het resultaat in HL is opgeslagen.

Operatiecode:

76	543	210	bitnummer.
00	ss1	001	byte1.

Voor ss moeten twee bits worden ingevuld die in onderstaande tabel te vinden zijn:

ss	paar.
00	BC
01	DE
10	HL
11	SP

De instructievolgorde moet zijn:

LD HL,adres 16-bitsgetal a
ADD HL,adres 16-bitsgetal b
LD adres 16-bitsgetal s,HL

Dit is in overeenstemming met het optellen van twee achtbitsgetallen. Voor het optellen van getallen groter dan zestien bits is voor de vervolgoptelling de instructie

ADC HL,ss

Operatiecode:

76	543	210	bitnummer.
11	101	101	byte1.
01	ss1	010	byte2.

Zie voor ss bovenstaande tabel.

In plaats van het registerpaar HL kunnen ook de indexregisters als doelregister worden gebruikt. Ook in dat geval bevindt zich het resultaat, na de bewerking, in het desbetreffende indexregister.

ADD IX,pp

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
00	pp1	001	byte2.

De bits die voor pp moeten worden ingevuld, staan in de volgende tabel:

pp	paar
00	BC
01	DE
10	IX
11	SP

ADD IY,rr

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
00	rr1	001	byte2.

Voor rr dienen de bits uit de volgende tabel te worden ingevuld:

rr	paar
00	BC
01	DE
10	IY
11	SP

Een instructie ADC die een indexregister als doelregister heeft, bestaat niet.

Uiteraard is ook een zestienbits aftrekking mogelijk. Hiervoor is slechts één instructie aanwezig, SBC. Nu moet vooraf, dus voordat de instructie SBC wordt toegepast, het Carrybit C een bepaalde waarde worden gegeven. De instructie SBC bepaalt van de aftrekker het twee-complement zonder hierbij het Carrybit te betrekken, zodat het Carrybit bij de eerste aftrekking eerst nul gemaakt moet worden. In de paragraaf 6.5. zal worden aangegeven hoe dat het beste kan gebeuren. Bij een vervolgaftrekking wordt de waarde van het Carrybit bepaald door de vorige bewerking.

SBC HL,ss

Bij deze bewerking bevindt het aftrektal zich in het doelregisterpaar HL. Dit aftrektal gaat verloren omdat het wordt vervangen door het resultaat.

Operatiecode:

76	543	210	bitnummer.
11	101	101	byte1.
01	ss0	010	byte2.

Zie voor ss de tabel die bij ADD HL,ss is gegeven.

Ook met zestienbitsregisters kan een teller worden gevormd:

INC ss

Operatiecode: 00 ss0 011.

INC IX

Operatiecode: 11011101 00100011 (DD23H).

INC IY

Operatiecode: 11111101 00100011 (FD23H).

DEC ss

Operatiecode 00 ss1 011.

DEC IX

Operatiecode: 11011101 00101011 (DD2BH).

DEC IY

Operatiecode: 11111101 00101011 (FD2BH).

De zestienbitsrekeninstructies zijn samengevat in lijst 7.

DESTINATION		SOURCE BC	 DE	 HL	 SP	 IX	 IY
'ADD'	HL	09	19	29	39		
'ADD'	IX	DD 09	DD 19		DD 39	DD 29	
'ADD'	IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC.		03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

Lijst 7. Zestienbitsrekeninstructies.

6.4. De vergelijkingsinstructies.

Het kan nogal eens gebeuren dat een voorwaardelijke sprong in een programma afhankelijk is van de grootte van een getal in een register. De sprong wordt dan uitgevoerd als het desbetreffende getal een bepaalde waarde heeft bereikt. Dit laatste moet dan gecontroleerd worden, hetgeen kan geschieden door vergelijking van het getal met een ander getal van de desbetreffende waarde. Deze vergelijking kan plaatsvinden door de twee getallen van elkaar af te trekken. Het nadeel is echter dat bij het uitvoeren van een SUB of een SBC instructie een van de getallen verloren gaat. Passen we echter de vergelijkingsinstructie CP (compare) toe, dan wordt wel een aftrekking uitgevoerd tussen een getal in de Accu en een getal dat zich in een geheugenlocatie bevindt, maar het resultaat van de aftrekking wordt niet geregistreerd, zodat het getal in de Accu niet verloren gaat. Belangrijk is echter dat de waarde van de FLAGS afhankelijk wordt gesteld van het resultaat van de aftrekking. Alle FLAGS worden beïnvloed zoals bij de SUB en SBC instructies. De voorwaarde die bij een sprongopdracht kan worden gesteld heeft steeds betrekking op de waarde van een van de FLAGS. Het is dus belangrijk de waarde van de FLAGS te kennen bij de diverse mogelijkheden die de waarden van de te vergelijken getallen ten opzichte van elkaar hebben. Deze mogelijkheden zijn:

a. Is kleiner dan.
b. Is gelijk aan.
c. Is groter dan.

Nu zou alles betrekkelijk eenvoudig zijn als beide getallen een gelijk teken hebben, dus beide positief of negatief zijn. Het is echter niet uit te sluiten dat de getallen verschillend van teken zijn. Zo is het getal 5 groter dan -1, maar als we de vergelijking gaan uitvoeren, dan krijgen de FLAGS een andere waarde dan bij de vergelijking tussen 5 en 1. Daarom moeten alle mogelijkheden worden onderzocht. We gebruiken daarbij de getallen a en r, r van referentie. Het getal a kan daarbij een veranderlijke waarde hebben. Het getal r is een constante waarmee a vergeleken moet worden.

Getal a is kleiner dan getal r, a < r.

De getallen a en r zijn beide positief.

a = 01011001 r = 01101011

00000001	Carrybit.
01011001	getal a.
10010100	NOT(r).
11101110	verschil v.

C = 0, S = 1, Z = 0, V = 0 (p).

De vergelijking wordt geheel volgens de regels van het aftrekken uitgevoerd. De instructie CP voert alle handelingen zelf uit.
Hetzelfde resultaat van de FLAGS wordt verkregen als beide getallen negatief zijn.
De getallen zijn tegengesteld van teken, a is negatief, r is positief.

a = 10010010 r = 01010010

00000001	Carrybit.
10010010	getal a.
10101101	NOT(r).
01000000	verschil v.

C = 1, S = 0, Z = 0, V = 1 (q).

Samenvatting a < r
C = 0, S = 1, Z = 0, V = 0 (p) of
C = 1, S = 0, Z = 0, V = 1 (q).

Zowel bij p als bij q is S ongelijk aan V zodat S ∀ V = 1.

Getal a is gelijk aan getal r, a = r.

a = r = 01101011 (positief).

00000001	Carrybit.
01101011	getal a.
10010100	NOT(r).
00000000	verschil v.

C = 1, S = 0, Z = 1, V = 0.

Hetzelfde resultaat wordt verkregen als a en r beide negatief zijn.
Nu is S gelijk aan V, zodat S ∀ V = 0

Getal a is groter dan getal r, a < r.

De getallen a en r zijn beide positief.

$a = 01101011$ $r = 01011001$

00000001	Carrybit.
01101011	getal a.
10100110	NOT(r).
00010010	verschil v.

$C = 1$, $S = 0$, $Z = 0$, $V = 0$ (t).

Hetzelfde resultaat van de FLAGS wordt verkregen als beide getallen negatief zijn.

De getallen zijn tegengesteld van teken, a is positief, r is negatief.

$a = 01011001$ $r = 10110110$

00000001	Carrybit.
01011001	getal a.
01001001	NOT(r).
10100011	verschil v.

$C = 0$, $S = 1$, $Z = 0$, $V = 1$ (u).

Samenvatting $a > r$:
$C = 1$, $S = 0$, $Z = 0$, $V = 0$ (t) of
$C = 0$, $S = 1$, $Z = 0$, $V = 1$ (u).

Nu is in beide gevallen S gelijk aan V, zodat ook hier geldt dat $S \forall V = 0$.

Bij het vergelijken van twee getallen zijn er vijf voorwaarden te stellen: $a < r$, $a < = r$, $a = r$, $a > = r$ en $a > r$. Indien het zeker is dat de getallen gelijk zijn van teken, dan volgen uit het voorgaande de volgende condities voor de FLAGS (dit zijn niet de enige mogelijkheden)

$a < r$: $C = 0$
$a < = r$: $C = 0 \vee Z = 1$
$a = r$: $Z = 1$
$a > = r$: $C = 1$
$a > r$: $C = 1 \wedge Z = 0$

Kunnen de getallen tegengesteld van teken zijn dan moeten de FLAGS voldoen aan

$a < r$: $S \forall V = 1$
$a < = r$: $Z \vee (S \forall Z) = 1$
$a = r$: $Z = 1$
$a > = r$: $S \forall V = 0$
$a > r$: $Z \vee (S \forall V) = 0$

De vergelijkingsinstructie wordt geschreven als

CP s

en hierbij kunnen dezelfde adresseermethoden worden toegepast als bij ADD s. Voor s kan dan ook r, n, (HL), (IX + d) of (IY + d) worden ingevuld. De operatiecode kan worden gevonden door in de operatiecode voor de overeenkomstige instructie voor ADD de cursief gedrukte bits te vervangen door *111*.
Belangrijk bij vergelijkingsinstructies is te weten welk getal de aftrekker en welk het aftrektal is. Bij de CP instructie bevindt het aftrektal (getal a) zich in de Accu en is de aftrekker (getal r) de operand die door s wordt geadresseerd.

De vergelijkingsinstructies vinden we in lijst 6.

6.5. Logische bewerkingen.
Er zijn drie logische bewerkingen die de computer kan uitvoeren: de "EN" ($\wedge$), de "OF" ($\vee$) en de "Exclusief OF" ($\forall$) bewerking. De instructies zijn:

AND s
OR s
XOR s

De bewerking vindt plaats tussen een getal in de Accu en de operand die door s wordt geadresseerd. Het resultaat bevindt zich na de bewerking in de Accu. De instructies hebben invloed op de waarde van de FLAGS. Hoe deze invloed is kan uit de instructieset worden afgelezen. In elk geval wordt na elke bewerking het Carrybit C 0. De computer past de instructies toe op de overeenkomstige bits van twee getallen. De voorbeelden van de logische bewerkingen in paragraaf 2.5. zijn volledig toepasbaar.

Met de "EN" bewerking is het mogelijk de waarde van een tetrade van een binair getal te isoleren. We maken hierbij gebruik van een *"masker"*. Stel we willen uit het getal 9B de lage tetrade isoleren. Daarvoor moet een "EN" bewerking worden uitgevoerd tussen dat getal en het getal 0F. Dit laatste is dan het masker.

10011011	getal a (9B).
00001111	masker m (0F).
00001011	$a \wedge m$, getal r (0B).

Door de "EN" bewerking tussen de getallen a en m bevat het resultaat r alleen de lage tetrade van getal a. Op gelijke wijze kan de hoge tetrade worden geïsoleerd:

10100111 getal b (A7).
11110000 masker m (F0).

10100000 b∧m, getal s (A0).

De "OF" bewerking maakt het mogelijk de getallen r en s samen te voegen:

10100000 getal s (A0).
00001011 getal r (0B).

10101011 s ∨ r, getal t (AB).

De instructievolgorde van de gehele bewerking moet zijn:

LD A,adres getal a.
AND 0F (masker)
LD C,A (getal r in reg. C)
LD A,adres getal b.
AND F0 (masker, getal s in A)
OR C (s∨ r in A)
LD adres getal t,A

Uit het voorgaande is gebleken dat het mogelijk is om een aantal bits van een getal in een register met een "EN" bewerking 0 te maken. Men noemt dat wel *"resetten"*. Met de "OF" bewerking kunnen een aantal bits 1 worden gemaakt, het *"setten"*.

76543210 bitnummer.
10100101 getal a.
00111100 masker m.

10111101 a∨m, nieuw getal a.

In dit voorbeeld zijn de bits 2 tot en met 5 van getal a 1 gemaakt, ook al waren ze reeds 1. Deze methode geeft de zekerheid dat alle gewenste bits worden geset, onafhankelijk van de waarde die de bits voordien hadden. De andere bits van het getal hebben nog hun oorspronkelijke waarde, hetgeen een voorwaarde is. Uiteraard bevindt het resultaat zich in de Accu en kan eventueel naar de geheugenplaats worden geschreven waar zich het getal a eerst bevond. Deze methode kan ook worden toegepast op een enkel bit van een getal. Dit is echter niet nodig omdat hiervoor speciale instructies zijn. De instructies AND en OR kunnen worden gebruikt voor het resetten (0 maken) van het Carrybit. Hiervoor is een logische bewerking nodig tussen een willekeurig getal in de Accu en zichzelf. Het getal in de Accu verandert daarbij niet. Als voorbeeld een "EN" bewerking.

10011011 getal a.
10011011 getal a.

10011011 a ∧ a, resultaat: getal a.

C=0.

De instructie die hierbij is gebruikt luidt:

AND A

Deze methode is gemakkelijk als twee getallen moeten worden afgetrokken die langer zijn dan zestien bits (paragraaf 6.3.). De instructievolgorde zou dan kunnen zijn:

LD HL,adres lage deel getal a.
AND A (Carrybit 0)
SBC HL,BC (lage deel getal b in BC)
LD adres lage deel verschil,HL
LD HL,adres hoge deel getal a.
SBC HL,DE (hoge deel getal b in DE)
LD adres hoge deel verschil,HL

Hiermee is de aftrekking $v=a-b$ tot stand gebracht. De instructie AND A wordt slechts voor de eerste stap van de aftrekking gebruikt. Daarna mag hij niet meer worden toegepast.

Bij de logische instructies kunnen dezelfde adresseermethoden worden toegepast als bij de instructie ADD. Voor s in

AND s

kan dan ook r, n, (HL), (IX+d) of (IY+d) worden ingevuld. De operatiecode kan worden gevonden door in de operatiecode voor de overeenkomstige instructie van ADD de cursief gedrukte bits te vervangen door 100. Op overeenkomstige wijze kunnen de operatiecodes voor

OR s

worden samengesteld. Nu moeten de cursief gedrukte bits worden vervangen door 110.

XOR s

Bij deze instructie moeten de cursief gedrukte bits worden vervangen door 101.

De logische instructies zijn samengevat in lijst 6.

6.6. De hulpinstructies.

De hulpinstructies worden gebruikt ter ondersteuning van de reken- en logische instructies. Eén van deze instructies is

NEG

Operatiecode: 11101101 01000100 (ED44H).

Hiermee kan het twee-complement van een getal in de Accu worden gevonden. De werkwijze komt overeen met wat in paragraaf 2.2. over negatieve getallen is vermeld. Het resultaat van de bewerking vinden we in de Accu, zodat het originele getal daarin verloren is gegaan. Een andere hulpinstructie die betrekking heeft op de rekeninstructies is

SCF

Operatiecode: 00110111 (37H).

Het doel van deze instructie is om de Carryflag 1 te maken (Set Carry Flag). Er is geen instructie om de Carryflag 0 te maken. In de vorige paragraaf is gebleken dat hiervoor AND A gebruikt kan worden. Wel is er nog de instructie

CCF

Operatiecode: 00111111 (3FH).

Hiermee wordt de waarde van de Carryflag tegengesteld aan de waarde die hij had (Complement Carry Flag). Een instructie die nodig is bij het rekenen in de BCD-code is

DAA

Decimal Adjust Accu.
Operatiecode: 00100111 (27H).

Deze instructie zorgt dat de correcties worden toegepast die de uitkomst van een binaire berekening omzet tot getallen in de BCD-code. De correcties voor het optellen vindt u in lijst 2 van paragraaf 2.4. De voorbeelden van het optellen die in deze paragraaf zijn gegeven kunnen zonder meer door de processor worden uitgevoerd. Voor het optellen van twee getallen a en b van elk twee tetraden kunnen de volgende instructies worden gebruikt:

```
LD A,adres a
ADD A,adres b
DAA
LD adres s,A
```

In eerste instantie is de optelling binair, uitgevoerd door ADD, zonder Carrybit. De instructie DAA die hierop volgt zorgt voor de juiste correcties. Het optellen van twee getallen a en b van elke vier tetraden moet, zoals bij gewoon binair optellen, in twee stappen gebeuren. De volgende instructievolgorde is hierbij nodig:

```
LD A,adres aL
ADD A,adres bL
DAA
LD adres sL,A
LD A,adres aH
ADC A,adres bH
DAA
LD adres sH,A
```

Het eerste deel (de eerste stap van de optelling) brengt niets nieuws. Bij de tweede stap moet de instructie ADC worden gebruikt omdat nu weer rekening moet worden gehouden met het optreden van een eventuele Carry. Opnieuw moet na de (binaire) berekening door ADC de instructie DAA worden gebruikt voor de juiste correcties. De instructie DAA kan zowel worden gebruikt na ADD en ADC als na INC, zodat ook een teller in de BCD-code kan worden gemaakt.

In paragraaf 2.4. is vermeld dat het aftrekken van twee getallen in de BCD-code door de processor op zijn geheel eigen wijze gebeurt. Deze wijze zal nu worden toegelicht.

Evenals bij het optellen wordt de aftrekking eerst binair uitgevoerd. Daarna moet weer een correctie volgen om het verkregen binaire resultaat om te zetten tot een getal in de BCD- code. Deze correctie verschilt t.o.v. die van de optelling. Allereerst moet de processor daarom weten welke bewerking zojuist heeft plaatsgevonden, een optelling of een aftrekking. Daarna kan hij vaststellen welke correctie moet worden gegeven, een correctie na een optelling dan wel een correctie na een aftrekking.

Hiervoor is in het FLAG-register de FLAG N. Deze FLAG wordt 0 bij een optelling (ADD, ADC en INC) en 1 bij een aftrekking (SUB, SBC DEC en NEG). Aan de N-FLAG kan de processor dus vaststellen of het een correctie voor een optelling dan wel voor een aftrekking moet zijn. De correctie die bij een optelling moet worden gegeven kan zowel van de waarde van de FLAGS H en C als van de grootte van de desbetreffende tetraden afhangen. Bij een aftrekking hangt de correctie echter uitsluitend van de waarde van H en C af. Het volgende voorbeeld is voor de aftrekking

82 getal a.
15 getal b.
——
67 verschil v.

1000 0010 getal a.
1110 1010 NOT(b).
0000 0001 Carrybit.
—— ——
0110 1101 H=0, C=1
0000 1010 correctie.
—— ——
0110 0111 Verschil v, C=1.

De aftrekking van de getallen a en b vindt plaats in het twee-complementsysteem en brengt dus niet veel nieuws. Bij deze aftrekking wordt H 0 en C 1. Aan de hand hiervan wordt bepaald dat de lage tetrade een correctie nodig heeft (H=0) en de hoge tetrade niet (C=1). Dit is tegengesteld aan wat voor een optelling gebruikelijk is. De grootte van de correctie is 1010 (AH). Dit is het twee-complement van de correctie die bij het optellen wordt gegeven:

0001 Carrybit.
1001 NOT(6 bin).
——
1010 twee-complement van 6.

Het optellen van de correctie bij de lage tetrade heeft een overdracht tot gevolg van de lage tetrade naar de hoge tetrade (van het vierde naar het vijfde bit). Deze overdracht wordt bij het optellen van de correctie *niet* uitgevoerd. Dit is overigens ook niet het geval als de correcties voor een optelling worden gelezen uit lijst 2. in paragraaf 2.4. In lijst 8 worden de correcties gegeven die na een aftrekking moeten worden uitgevoerd op een getal van twee tetraden, waarbij n de hoge en m de lage tetrade is.

C	H	Resultaat
1	1	n,m C=1
1	0	n,m+A C=1
0	1	n+A,m C=0
1	1	n+A,m+A C=0

Lijst 8. Correcties bij een BCD-aftrekking.

De instructievolgorde voor het voorgaande voorbeeld luidt:

```
LD A,getal a
SUB adres b
DAA
LD adres v,A
```

Het aftrekken van grotere getallen moet weer in meerdere stappen gebeuren.

7125 getal a.
3687 getal b.
——
3438 verschil v.

1e stap:

0010 0101 getal aL.
0111 1000 NOT(bL).
0000 0001 Carrybit.
—— ——
1001 1110 H=0, C=0.
1010 1010 correctie.
—— ——
0011 1000 verschil vL, C=0.

2e stap:

0111 0001 getal aH.
1100 1001 NOT(bH).
0000 0000 carrybit.
—— ——
0011 1010 H=0, C=1.
0000 1010 correctie.
—— ——
0011 0100 verschil vH.

Verschil v: 00110100 00111000.

In het eerste voorbeeld zijn na de binaire aftrekking H 0 en C 0, zodat beide tetraden met 1010 gecorrigeerd moeten worden. Ook nu moet de overdracht van de lage naar de hoge tetrade achterwege blijven. Er moet geleend worden van het hoge deel

van getal a, vandaar dat C = 0 in het resultaat van de berekening (zie lijst 8). De tweede stap verloopt overeenkomstig aan de eerste. Merk op dat het carrybit dat bij de aftrekking is gebruikt nu de waarde heeft die hij bij de eerste stap heeft gekregen (0). De instructievolgorde is

```
LD A,adres aL
SUB adres bL
DAA
LD adres vL,A
LD A,adres aH
SBC adres bH
DAA
LD adres vH,A
```

Behalve bij SUB, SBC en DEC kan DAA ook nog worden gebruikt bij de instructie NEG. Het is daardoor mogelijk het 10-complement te bepalen van een BCD getal. Stel dat van het getal a 01100011_{BCD} het 10-complement moet worden bepaald. De werkwijze is als volgt

```
0000  0001 Carrybit.
1001  1100 NOT(a)
----  ----
1001  1101 2-complement a.
1010  1010 correctie.
----  ----
0100  0011
```

01000011_{BCD} is het 10-complement van 01100011_{BCD}.

Instructievolgorde:

```
LD A adres get
NEG
DAA
```

De volgende hulpinstructie kan bij de logische bewerkingen worden gebruikt:

CPL

Complement Accumulator.
Operatiecode: 00111111 (02FH)

Deze instructie bepaalt het complement van het getal in de Accu (een 1 wordt 0 en een 0 wordt 1) en plaatst het resultaat weer terug in de Accu. De oorspronkelijke inhoud hiervan gaat dus verloren. De hulpinstructies zijn samengevat in lijst 9.

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

Lijst 9. Hulpinstructies voor rekenkundige en logische bewerkingen.

6.7. Rotatie- en schuifinstructies.

De rotatie- en schuifinstructies verplaatsen elk bit in het geadresseerde register een plaats naar rechts of links, afhankelijk van de aard van de instructie. Hierbij gaat geen enkel bit verloren. Ook het bit dat "uit het register" wordt geschoven (bij schuiven naar links bit 7 bijvoorbeeld) wordt op de een of andere manier bewaard. Aan de andere kant van het register wordt een bit ingeschoven. Dit kan bijvoorbeeld 0 zijn maar ook de inhoud van de Carryflag. De rotatie-instructies zijn in principe schuifinstructies. Omdat het hierbij mogelijk is een uitgeschoven bit aan de andere kant van het register weer binnen te schuiven wordt wel van rotatie gesproken. Een dergelijke rotatie-instructie zien we in figuur 52. Deze wordt aangeduid met RLC, Rotate Left Circulair. De zeven bits van het getekende register worden allemaal gelijktijdig naar links geschoven. Daardoor schuift bit 7 als het ware uit het register. De Carryflag krijgt de waarde

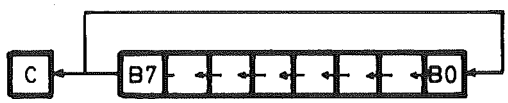


Fig. 52. Rotatie naar links.

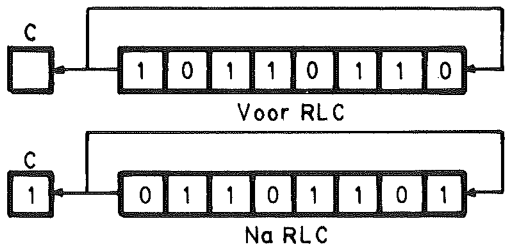


Fig. 53. Voorbeeld van een rotatie naar links.

van dit bit, evenals bit 0. Als de instructie steeds opnieuw op het register wordt toegepast dan roteren de bits in het register. In figuur 53 is de situatie van het register getekend voor en na de werking van de instructie. Als de instructie acht keer is gebruikt, dan heeft de Carryflag achtereenvolgens de waarde gehad van alle bits van het getal in het register.

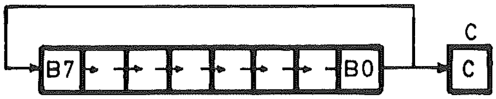


Fig. 54. Rotatie naar rechts.

Een rotatie naar rechts is ook mogelijk. In dat geval schuift bit 0 uit het register in de Carryflag en in het bit 7 (fig. 54). Hoe vaak de instructie ook wordt toegepast, het getal in het register gaat in principe niet verloren. Dat is anders met de schuifinstructie uit figuur 55. Deze wordt met SLA aangegeven, Shift Left Arithmetic. Bit 7 schuift uit het register in de Carryflag terwijl bit 0 nul wordt. Stel dat de inhoud van het register en de Carryflag voor het schuiven naar links gelijk is aan 01011101, 93 decimaal. Na het schuiven is de inhoud dan 10111010, 186 decimaal. Dat betekent dat een getal dat in een register de bewerking SLA ondergaat, met 2 wordt vermenigvuldigd.

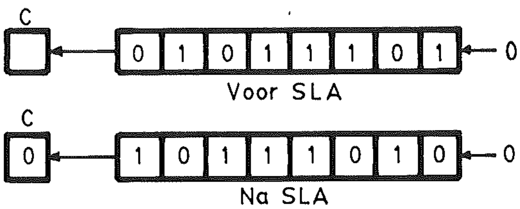


Fig. 55. Schuiven naar links.

In principe kan deze instructie als een rekeninstructie worden beschouwd. Het is heel goed mogelijk een dergelijke bewerking (vermenigvuldigen met 2) te gebruiken bij grotere getallen dan achtbits. Dan moet SLA worden gecombineerd met de instructie RL (Rotate Left) van figuur 56. Bij deze instructie is de Carryflag in de ''rotatieketen'' opgenomen.

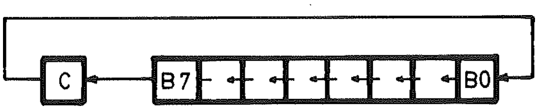


Fig. 56. Rotatie naar links via de Carryflag.

Stel dat de bewerking op twee registers moet worden toegepast. Op de lage byte van het getal in de registers wordt de bewerking SLA losgelaten. Bit 0 wordt daardoor 0 en bit 7 wordt in de Carryflag geschoven. Daarna wordt RL toegepast op de hoge byte van het getal. Hiervan krijgt bit 0 de waarde van de Carryflag, zodat uiteindelijk bit 7 van de lage byte terechtgekomen is in bit 0 van de hoge byte. Verder zijn alle bits van de hoge byte een plaats naar links geschoven en is bit 7 hiervan in de Carryflag terecht gekomen. Door deze twee instructies zijn dus alle bits van de twee registers naar links geschoven, zonder dat een bit verloren is gegaan (figuur 57). Deze keten kan ook op meerdere registers worden toegepast. Dan moet bij elk extra register de instructie RL worden gebruikt.

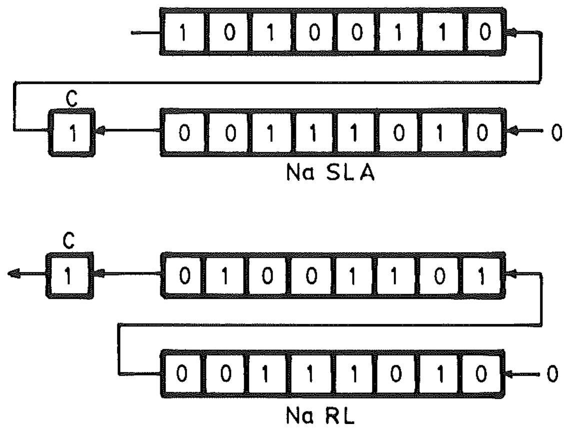


Fig. 57. Naar links schuiven door twee registers.

Uiteraard is er ook een instructie waarmee delen door 2 mogelijk is. Een voorwaarde daarbij is dat de waarde van bit 7, het tekenbit, niet verloren gaat. De instructie die dat garandeert is SRA, Shift Right Arithmetic (fig. 58). Wordt deze instructie op een register toegepast, dan schuiven alle bits

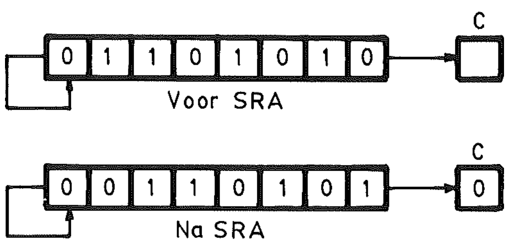


Fig. 58. Schuiven naar rechts met behoud van teken.

naar rechts, waarbij bit 7 zijn waarde blijft behouden. Na het schuiven hebben de bits 7 en 6 dus dezelfde waarde. Bit 0 schuift in de Carryflag. Als een positief getal in het register is, dan is bit 7 nul. Stel het getal is 01101010, 106 decimaal. Na het schuiven is de registerinhoud 00110101, 53 decimaal. Hier heeft dus de deling door 2 plaatsgevonden. Het bit in de Carryflag kan als de rest van de deling worden beschouwd.
Bij een negatief getal is het tekenbit 1. Stel het getal in het register is 10110110, −74. Na het schuiven is dat 11011011, −37.

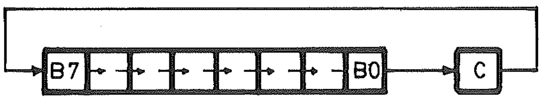


Fig. 59. Rotatie naar rechts via de Carryflag.

Om de bewerking "delen" ook op meerdere registers te kunnen toepassen is de instructie RR (Rotate Right). Hierbij schuiven de bits van het register naar rechts, waarbij bit 7 de waarde van de Carryflag krijgt en bit 0 uiteindelijk in de Carryflag schuift (figuur 59). Op het register met de hoogste byte moet de instructie SRA worden toegepast terwijl bij de andere registers, na elkaar van hoog naar laag, de instructie RR moet worden gebruikt. Verder werkt het geheel in overeenstemming met de bewerking voor vermenigvuldigen.
Voor het delen van *positieve* getallen door 2 kan in plaats van SRA ook de instructie SRL (Shift Right Logical) worden toegepast. Hierbij wordt bij bit 7 een 0 ingeschoven bij het naar rechts gaan van de bits. Bit 0 schuift in de Carryflag (figuur 60).

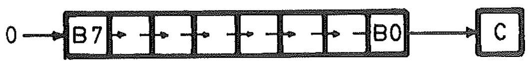


Fig. 60. Schuiven naar rechts voor positieve getallen.

Voor het schuiven naar links van een hele tetrade dient de instructie RLD (Rotate Left Digit). Hierbij neemt de lage tetrade van het geadresseerde register de plaats in van de tweede en de tweede tetrade neemt de plaats in van de lage terade in de Accu.

Deze laatste zijn oorspronkelijke inhoud af aan de tetrade van het geadresseerde register (figuur 61). Het is dus een rotatie waarbij geen van de tetraden verloren gaat. De tegenovergestelde richting is ook mogelijk met de instructie RRD (Rotate Right Digit). Deze instructie is afgebeeld in figuur 62.

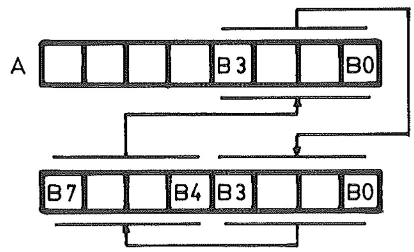


Fig. 61. Schuiven naar links van een tetrade.

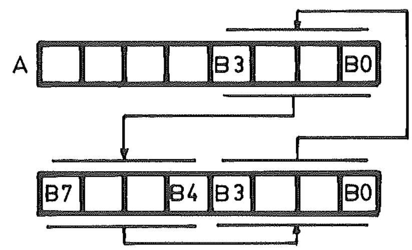


Fig. 62. Schuiven naar rechts van een tetrade.

Het adres van het register waarvan de tetraden zullen schuiven, wordt gevonden in de inhoud van het processorregisterpaar HL.

RLD

Operatiecode:

76	543	210	bitnummer.
11	101	101	(EDH) byte1.
01	101	111	(6FH) byte2.

RRD

Operatiecode:

76	543	210	bitnummer.
11	101	101	(EDH) byte1.
01	100	111	(67H) byte1.

Alle vier de rotatie-instructies kunnen toegepast worden op de inhoud van de Accu.

RLCA

Operatiecode: 00000111 (07H).

RLA

Operatiecode 00010111 (17H).

RRCA

Operatiecode: 00001111 (0FH).

RRA

Operatiecode: 00011111 (1FH).

Andere adresseringsmogelijkheden zijn register, register indirect en indexed. Voor de instructie RLC kunnen de operatiecoden worden samengesteld.

RLC r

Operatiecode:

76	543	210	bitnummer.
11	001	011	byte1.
00	*000*	r..	byte2.

De drie bits die hierin voor r moeten worden ingevuld kunnen worden gelezen uit de volgende tabel:

r..	reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

RLC (HL)

Operatiecode:

76	543	210	bitnummer.
11	001	011	byte1.
00	*000*	110	byte2.

RLC (IX + d)

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
11	001	011	byte2.
d.	...	...	byte3.
00	*000*	110	byte4.

RLC (IY + d)

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
11	001	011	byte2.
d.	...	...	byte3.
00	*000*	110	byte4.

RL s

Bij alle rotatie-instructies zijn dezelfde adresseringsmogelijkheden. Voor s kunt u dan ook lezen: r, (HL), (IX + d) en (IY + d). De operatiecoden zijn te vinden door in de overeenkomstige operatiecoden voor RLC de cursief gedrukte bits te vervangen door 010.

RRC s

Voor deze instructie geldt hetzelfde als de vorige. De drie cursief gedrukte bits moeten echter vervangen worden door 001.

RR s

Voor de operatiecoden moeten de cursief gedrukte bits vervangen worden door 011.

De schuifinstructies zijn niet toe te passen op de inhoud van de Accu. De andere adresseringsmogelijkheden zijn echter gelijk aan die van de rotatie-instructies. De operatiecoden zijn samen te stellen door in de overeenkomstige operatiecoden voor RLC de cursief gedrukte bits te vervangen.

SLA s

Hierbij moeten de bits 100 worden gebruikt.

SRA s

Nu moeten de bits 101 worden ingevuld.

SRL s

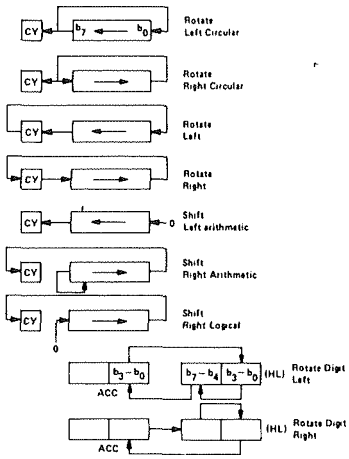
De bits 111 moeten worden ingevuld.

De rotatie- en schuifinstructies zijn samengevat in lijst 10.

Source and Destination

TYPE OF ROTATE OR SHIFT		A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
	'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
	'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
	'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
	'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
	'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
	'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
	'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
	'RLD'								ED 6F		
	'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F



Lijst 10. Rotatie- en schuifinstructies.

6.8. De bitmanipulatiegroep.

In paragraaf 6.5. is beschreven op welke manier een aantal bits van een getal in een register, door middel van een masker, een bepaalde waarde kan worden gegeven. Uiteraard kan op die manier ook een enkel bit een waarde worden toegekend. Veel eenvoudiger gaat dat echter met een instructie uit de bitmanipulatiegroep. Door middel van een enkele instructie is het mogelijk om een bepaald bit van een register te setten (''1'' maken) of te resetten (''0'' maken) *zonder dat daarbij de andere bits in het register van waarde veranderen*. Ook zijn er instructies waarmee de waarde van een bepaald bit kan worden vastgesteld, zonder dat het wordt veranderd. Om met deze laatste te beginnen, hiervoor dient de instructie

BIT b

Hierin is b het nummer (van 0 tot en met 7) van het desbetreffende bit. Wordt deze instructie op een bepaald register toegepast, dan krijgt de FLAG Z de *tegengestelde waarde* van het desbetreffende bit. Op deze instructie zijn de adresseermethoden register, register indirect en indexed toepasbaar. De manier waarop de waarde van een bit wordt bepaald is geheel in overeenstemming met de methode die in paragraaf 6.5. is beschreven, dus met gebruik van een masker. Stel dat van een bepaald getal (a) in een register de waarde van bit 4 moet worden bepaald. Het volgende vindt dan plaats:

76543210 bitnummer.
10010110 getal a.
00010000 masker m.
____
00010000 $a \wedge m$, $Z = 0$.

Het resultaat van de EN-bewerking van getal a met het masker m is *niet* 0. Dat betekent dat de Z-FLAG 0 is. Dit is dus in tegenstelling met de waarde van het bit 4.

76543210 bitnummer.
10000110 getal a.
00010000 masker m.
____
00000000 $a \wedge m$, $Z = 1$.

Bij dit voorbeeld is het resultaat van de bewerking 0, zodat de Z-FLAG 1 is, ook alweer tegengesteld aan de waarde van bit 4. De gehele bewerking wordt door de instructie BIT b veroorzaakt.

BIT b,r

Operatiecode:

76	543	210	bitnummer.
11	001	011	byte1.
01	b..	r..	byte2.

In de operatiecode moeten voor b, afhankelijk van het te testen bit, drie bits worden ingevuld die te lezen zijn uit de volgende tabel:

b.. bitnummer.
000 0
001 1
010 2
011 3
100 4
101 5
110 6
111 7

Deze tabel wordt voor alle operatiecoden in deze paragraaf gebruikt. Voor r moeten drie bits worden ingevuld die afhankelijk zijn van het gekozen processorregister en gevonden worden in onderstaande tabel:

r... reg.
000 B
001 C
010 D
011 E
100 H
101 L
111 A

Moet het bit 4 getest worden van processorregister C, dan luidt de instructie

BIT 4,C

Operatiecode: 11001011 01100001 (CB61H).

BIT b,(HL)

Operatiecode:

76	543	210	bitnummer.
11	001	011	byte1.
01	b..	110	byte2.

BIT b,(IX+d)

Operatiecode:

76	543	210	bitnummer.
11	011	101	byte1.
11	001	011	byte2.
d.	...	...	byte3.
01	b..	110	byte4.

BIT b.(IY+d)

Operatiecode:

76	543	210	bitnummer.
11	111	101	byte1.
11	001	011	byte2.
d.	...	...	byte3.
01	b..	110	byte4.

		REGISTER ADDRESSING							REG. INDIR.	INDEXED	
	BIT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	Cd 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	GB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

Lijst 11. De bitmanipulatiegroep.

Voor het setten van een bepaald bit kan de volgende instructie worden gebruikt:

SET b,s

Op deze instructie zijn dezelfde adresseermethoden toepasbaar als bij BIT b en voor s kan worden gelezen: r, (HL), (IX + d) of (IY + d). De desbetreffende operatiecode kan worden gevormd door in de overeenkomstige operatiecoden voor BIT b de cursief gedrukte bits te vervangen door 11.

Voor het resetten van een bepaald bit kan worden gebruikt:

RES b,s

Hierbij geldt hetzelfde als bij SET b,s. De cursief gedrukte bits in de operatiecode moeten worden vervangen door 10.

De bitmanipulatie-instructies zijn samengevat in lijst 11.

6.9. Sprongopdrachten.

Reeds in paragraaf 1.4. heeft u kennis kunnen maken met sprongopdrachten. Hierbij zijn twee mogelijkheden naar voren gekomen, de *onvoorwaardelijke* of *ongeconditioneerde* sprong (fig. 5) en de *voorwaardelijke* of *geconditioneerde* sprong. Daar bleek ook dat de toepassing van een onvoorwaardelijke sprong niet altijd zo'n beste oplossing is. Elke sprongopdracht, in de vorm zoals die in deze paragraaf is genoemd (JUMP), doet de inhoud van de programmateller veranderen zodat op een nieuw adres de sequentie wordt voortgezet. Bij een onvoorwaardelijke sprong terug in het programma wordt steeds hetzelfde programmadeel doorlopen en komen we ook steeds weer dezelfde sprongopdracht tegen. Als er verder niets aan gedaan wordt zal de computer ongelimiteerd hetzelfde programmadeel blijven doorlopen. De onvoorwaardelijke sprong maakt het ook mogelijk om van hot naar her door de geheugenruimte te springen, hetgeen de overzichtelijkheid van een programma nu niet bepaald bevordert. In principe moet een onvoorwaardelijke sprong dus niet worden toegepast. Toch laat deze zich niet altijd vermijden. Bij de programmadelen in de figuren 9 tot en met 11 moeten we aan het eind van het progammadeel, na het uitvoeren van een bepaalde instructie, uiteindelijk steeds weer op hetzelfde adres uitkomen en dat kan alleen maar met een ongeconditioneerde sprongopdracht. Verder zijn we de noodzaak van een ongeconditioneerde sprong ook nog tegengekomen in paragraaf 4.6. Een ongeconditioneerde sprong waarbij de immediate extended addressing mode is toegepast is

JP nn

De twee bytes van het adres dat in de programmateller moet worden geladen volgen hier onmiddellijk op de operatiecode.

Operatiecode:

76	543	210	bitnummer.
11	000	011	(C3H) byte1.
n.	...	...	byte2, adres laag.
n.	...	...	byte3, adres hoog.

De werking van een JP nn opdracht, waarvan de operatiecode C3 zich op het adres 9100 bevindt en waarvan het sprongadres 9158 is, is verklaard bij figuur 35.

Andere adresseermogelijkheden zijn:

JP (HL)

Operatiecode: 11101001 (E9).

JP (IX)

Operatiecode: 11011101 11101001 (DDE9H).

JP (IY)

Operatiecode: 11111101 11101001 (FDE9H).

JR e

Operatiecode:

76	543	210	bitnummer.
00	011	000	(18H) byte1.
e.	...	...	byte2, relatief adres.

Bij de jumpinstructie JR e wordt de Relative Addressing mode toegepast (paragraaf 4.10). Het getal e, de tweede byte van de instructie, wordt wel het *"relatieve adres"* genoemd. Deze byte geeft niet direct het sprongadres aan maar vermeldt in principe het aantal geheugenplaatsen dat vooruit of achteruit moet worden gesprongen. Het uitgangsadres is het adres van de operatiecode *plus*

twee. Dat komt omdat de processor eerst de twee bytes uit het geheugen moet ophalen, waardoor de programmateller twee keer is verhoogd. Figuur 36 toont een voorwaartse relatieve sprong. De operatiecode staat hierbij op het adres 9100. Na het binnenhalen van de twee bytes van de instructie is de inhoud van de programmateller 9102, zodat het sprongadres 9102 + e = 9102 + 56 = 9158 wordt. De maximale voorwaartse sprong bedraagt 127 geheugenplaatsen. Dit is het grootste positieve getal dat in een achtbitsregister past (01111111). Nemen we een negatief getal voor e, dan wordt dit omgezet in het twee-complementsysteem en in byte2 geplaatst.

De optelling vindt geheel plaats zoals dat voor het twee-complementsysteem is voorgeschreven, zodat de nieuwe waarde van de programmateller kleiner zal zijn dan het uitgangsadres. Het grootste negatieve getal dat in een achtbits register past is – 128 (10000000) en dat is dan ook het grootste aantal geheugenplaatsen dat bij een sprong terug in het programma kan worden "overgeslagen". Zijn grotere sprongen dan + 127 en – 128 nodig, dan moet een van de andere adresseermogelijkheden worden gebruikt.

Ook met een voorwaardelijke sprong hebben we al eens kennisgemaakt. Ga na wat hierover bij de figuren 6 tot en met 11 is vermeld (paragraaf 1.4. en paragraaf 2.6.). Of een sprong moet plaatsvinden is steeds afhankelijk van het resultaat van een berekening. Hierdoor worden de FLAGS in het FLAG-register geset of gereset. De FLAGS Z, C, V en S kunnen als voorwaarde voor een sprong dienen. In de tabellen wordt V ook wel met P aangegeven (Parity). Dit geeft de keuze uit acht mogelijkheden. In de volgende gevallen wordt de sprong gemaakt:

Z = 0; NZ, Non Zero (het resultaat is niet nul).
Z = 1; Z, Zero (het resultaat is nul).
C = 0; NC, Non Carry.
C = 1; C, Carry.
V = 0; PO, Parity Odd.
V = 1; PE, Parity Even.
S = 0; P, sign Positive.
S = 1; M, sign Minus (ook: negative).

Wordt niet aan een voorwaarde voldaan, dan kan de sprong niet plaatsvinden en wordt het programma vervolgd met de eerstvolgende instructie.

Er zijn twee adresseermogelijkheden, immediate extended en relatief.

JP cc,nn

Voor cc kunnen de voorwaarden NZ, Z, NC, C, PO, PE, P of M worden ingevuld. De operatiecode kan worden samengesteld met

76	543	210	bitnummer.
11	cc.	010	byte1.
n.	...	...	byte2, adres laag.
n.	...	...	byte3, adres hoog.

Voor cc dienen drie bits te worden ingevuld die gevonden kunnen worden in de volgende tabel:

cc. voorwaarde.
000 NZ, Non Zero
001 Z, Zero
010 NC, Non Carry
011 C, Carry
100 PO, Parity Odd
101 PE, Parity Even
110 P, sign Positive
111 M, sign Minus

JR dd,e

Bij de Relative Addressing mode zijn slechts vier mogelijkheden. Voor dd kan ingevuld worden: NZ, N, NC of C. De operatiecode kan worden samengesteld met

76	543	210	bitnummer.
00	1dd	000	byte1.
e.	...	...	byte2, relatief adres.

De twee bits voor dd kunnen gevonden worden in de volgende tabel:

dd voorwaarde.
00 NZ, Non Zero
01 Z, Zero
10 NC, Non Carry
11 C, Carry

Als voorbeeld nemen we het stroomdiagram in fig. 6. Hier worden een aantal geheugenplaatsen 0 gemaakt. Zoals later zal blijken is het bij de Z80 niet mogelijk hiervoor de geheugenplaatsen 0001 tot en

met 0005 te gebruiken. We zullen hiervoor de geheugenplaatsen E801 tot en met E805 gebruiken.

De bedoeling is dat zolang nog niet alle geheugenplaatsen van de juiste inhoud zijn voorzien (0), een sprong terug in het programma wordt gemaakt. Deze sprong is hier afhankelijk van het resultaat van de bewerking "decrement", dat wil zeggen dat het resultaat van de bewerking niet nul moet zijn, wil de sprong plaatsvinden. Gekozen is voor de volgende instructievolgorde:

LD HL,E805 (1)
LD A,0 (2)
LD(HL),A (3)
DEC L (4)
JP NZ,adres (5)
Volgende instructie (6)

Om te beginnen wordt het registerpaar HL met het adres geladen dat als eerste moet worden bewerkt (1). Daarna wordt de inhoud van de Accu voorzien van de waarde die later in de vijf geheugenplaatsen moet worden geschreven (2). De derde instructie (3) voorziet eerst adres E805 van de waarde van de Accu. Dan wordt alleen het register L met 1 verminderd (4). Het adres in het registerpaar HL is nu E804. Omdat het resultaat van de DEC- instructie niet 0 is, wordt door JP NZ teruggesprongen naar regel (3) waar nu geheugenplaats E804 met 0 wordt geladen. Het programma gaat zo door totdat de geheugenplaats E801 is geladen. De inhoud van register L is nu 1 en de instructie DEC heeft daarna 0 als resultaat. Een sprong heeft nu niet plaats en het programma vervolgt met regel (6). Vergelijk de instructies met de blokken uit figuur 6. Laten we het programma aanvangen op het adres E000, dan moeten de achtereenvolgende geheugenplaatsen van de volgende codegetallen worden voorzien:

E000 21 05 E8; LD HL,E805 (1)
E003 3E 00; LD A,0 (2)
E005 77; LD (HL),A (3)
E006 2D; DEC L (4)
E007 C2 05 E0; JP NZ,E005 (5)
E00A (6)

Alle getallen zijn hexadecimaal. Elke regel geeft de volledige instructie, eerst het adres van de eerste byte daarvan en daarna de codegetallen. Merk op dat het adres in regel (1) en in regel (5) in de volgorde lage byte - hoge byte in de geheugenplaatsen is geladen.
In dit geval kan ook de Relative Addressing mode worden toegepast. Regel (5) wordt dan

JR NZ,e

Het relatieve adres e moet daarbij berekend worden. Eerst moet het uitgangsadres worden bepaald. De instructie begint op het adres E007. Het uitgangsadres is twee geheugenplaatsen verder: E009. Het sprongadres is E005. Het relatieve adres wordt steeds berekend door het getal van het uitgangsadres van dat van het sprongadres af te trekken, ook bij sprongen vooruit in het programma.

e = E005 – E009 = FFFC

Deze berekening kunt u door uw computer laten uitvoeren (A = &HE005 – &HE009:PRINT HEX$(A). U krijgt dan het antwoord direct in het hexadecimale codesysteem. Van het antwoord moet alleen de lage byte FC voor e worden ingevoerd. Het programma wordt dan in codegetallen:

E000 21 05 E8; LD HL,E805 (1)
E003 3E 00; LD A,0 (2)
E005 77; LD (HL),A (3)
E006 2D; DEC L (4)
E007 20 FC; JR NZ,FC (5)
E009 (6)

U ziet dat het uitgangsadres gelijk is aan het adres waarmee het programma wordt voortgezet.

Er is nog een instructie niet ter sprake gekomen:

DJ NZ,e

Deze instructie combineert DEC B en JR NZ,e. Het komt nogal eens voor dat een sprong moet worden gemaakt als de inhoud van een teller niet nul is. In ons vorige voorbeeld is de teller het register L. Bij de instructie DJ NZ.e wordt het register B als teller gebruikt. De inhoud hiervan wordt eerst met 1 verminderd. Is de inhoud daarna niet nul, dan wordt de sprong gemaakt waarvoor e het relatieve adres weergeeft. De operatiecode is:

76	543	210	bitnummer.
00	010	000	byte1.
e.	...	...	byte2, relatief adres.

			CONDITION									
			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'		(HL)	E9									
JUMP 'JP'	REG. INDIR.	(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

Lijst 12. De sprong- en subroutine-instructies.

De spronginstructies worden samengevat in lijst 12.

6.10. Blokverplaats- en blokvergelijkingsinstructies.

In de voorgaande paragraaf heeft u al kennis kunnen maken met een instructie die het werk van twee enkelvoudige instructies in zich gecombineerd heeft (DJ NZ,e). De Z80 kent nog een aantal instructies die het werk doen van meerdere enkelvoudige instructies. Stel dat de inhoud van een aaneengesloten aantal geheugenplaatsen (blok) moet worden verplaatst naar een aantal, ook aaneengesloten, geheugenplaatsen. Dit is een handeling die nogal eens voorkomt in bepaalde programma's. De volgende instructies zouden hiervoor kunnen worden gebruikt:

LD HL,E300; beginadres van de bron (1).
LD DE,E100; beginadres van het doel (2).
LD BC,0100; aantal te verplaatsen adressen (3).
LD A,(HL); haal data uit bronregister (4).
LD (DE),A; plaats data in doelregister (5).
INC HL; verhoog pointer van de bron (6).
INC DE; verhoog pointer van het doel (7).
DEC BC; verlaag de teller (8).
JP NZ,adres; sprong naar regel 4 (9).

Bij dit voorbeeld is de plaats van het bronregisterblok gesteld op E300-E3FF en de plaats van het doelregisterblok op E100-E1FF. Dat betekent dat de inhoud van 256 geheugenplaatsen moet worden verplaatst. Bij dit programma zijn twee *"pointers"* nodig, namelijk een waarde die het adres aangeeft waar de data moet worden opgehaald en een waarde die het adres aangeeft waar de data moet worden ingeschreven. De pointer van de bron wordt in HL geladen. Omdat we beginnen met het laagste adres van het blok is de pointer E300 (regel 1). De doelpointer wordt in DE geladen (regel 2). Een teller geeft het aantal van de te verplaatsen data. Hiervoor wordt BC gebruikt (regel 3). Nu kan het eigenlijke overbrengen van de data beginnen. Eerst wordt de data uit het bronregister in de Accu geladen (regel 4). Deze data wordt daarna in het doelregister geladen (regel 5). Hierna moeten de pointers voor de bron en het doel worden verhoogd voor het behandelen van de volgende geheugen-

plaatsen (regels 6 en 7). Steeds als data is overgebracht, wordt de "stand" bijgehouden door het verlagen van de teller. Zolang deze niet nul is wordt de werking herhaald vanaf regel 4. Wordt door DEC BC de teller nul, dan is de data overgebracht en kan de sprong JP NZ,e (regel 9) niet plaatshebben. Aan het einde van het programma staan de pointers op respectievelijk E400 en E200, een adres hoger dan het hoogste adres van het desbetreffende blok.
De Z80 kent een instructie die de werking van de regels 4 tot en met 8 in zijn geheel uitvoert:

LDI

LoaD and Increment.
Operatiecode: 11101101 10100000 (EDA0H).

Het programma wordt daarmee veel eenvoudiger:
LD HL,E300
LD DE,E100
LD BC,0100
LDI
JP PE,adres

Merk op dat JP-instructie op de laatste regel is veranderd van JP NZ,adres in JP PE,adres. Zolang de waarde van het registerpaar BC niet nul is wordt door LDI de P/V-FLAG geset. Wordt na DEC BC de inhoud van BC echter nul, dan wordt de P/V-FLAG gereset.Er is een instructie die nog verder gaat en de regels 4 tot en met 9 combineert:

LDIR

LoaD, Increment and Repeat.
Operatiecode: 11101101 10110000 (EDB0H).

De instructievolgorde is nu:

LD HL,E300
LD DE,E100
LD BC,0100
LDIR

Uiteraard zijn de ingevoerde waarden voor de pointers en de teller slechts voorbeelden. Het mag dan waar zijn dat LDIR een grotere vereenvoudiging geeft dan LDI, deze laatste instructie geeft de mogelijkheid nog meer instructies "in de lus" (het gedeelte dat zich steeds herhaalt) in te lassen. Het grootste getal dat het registerpaar BC kan bevatten is FFFFH. Daarom kan maximaal de inhoud van een 64 Kbyte geheugenblok verplaatst worden. Het kan gebeuren dat het doelregisterblok het bronregisterblok overlapt. In dat geval moet de pointer van het *doelregisterblok* steeds een *lagere* waarde hebben dan die van het bronregisterblok. Heeft de pointer van het doelregisterblok een hogere waarde dan die van het bronregisterblok, dan wordt op een gegeven moment een register met een waarde ingeschreven, nog voordat de oorspronkelijke waarde hiervan is uitgelezen door LD (HL),A. Werd in het vorige programma de data van laag naar hoog overgebracht, nu moet de data van hoog naar laag worden verplaatst.

LD HL,E1FF; hoogste adres bronregister (1).
LD DE,E3FF; hoogste adres doelregister (2).
LD BC,0100; teller (3).
LD A,(HL); data uit bronregister (4).
LD (DE),A; data in doelregister (5).
DEC HL; verlaag pointer bronregister (6).
DEC DE; verlaag pointer doelregister (7).
DEC BC; verlaag de teller (8).
JR NZ,e; sprong naar regel 4 (9).

Nu heeft de pointer van het doelregister steeds een hogere waarde dan die van het bronregister. Omdat de data van hoog naar laag wordt overgebracht kan dat bij een eventuele overlapping geen moeilijkheden geven. De regels 4 tot en met 8 worden gecombineerd in de instructie

LDD

LoaD and Decrement.
Operatiecode: 11101101 10101000 (EDA8H).

Voor de combinatie van de regels 4 tot en met 9:

LDDR

LoaD, Decrement and Repeat.
Operatiecode: 11101101 10111000 (ED88H).

De blokverplaatsingsinstructies zijn weergegeven in lijst 13.

Het gebeurt ook nogal eens dat in een bepaald geheugenblok moet worden gezocht naar een geheugenplaats waarin een bepaalde data is opgeslagen.

DESTINATION			SOURCE: REG. INDIR. (HL)	
	REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)◄—(HL) Inc HL & DE, Dec BC
			ED B0	'LDIR,' – Load (DE)◄—(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
			ED A8	'LDD' – Load (DE)◄—(HL) Dec HL & DE, Dec BC
			ED B8	'LDDR' – Load (DE)◄—(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Lijst 13. De blokverplaatsingsinstructies.

Stel dat in het geheugenblok E100-E1FF gekeken moet worden in welke geheugenplaats de data A3 is opgeslagen. Dat kan met het volgende programma:

LD HL,E100; beginadres pointer (1).
LD BC,0100; teller (2).
LD A,A3; te vergelijken data (3).
CP (HL); vergelijk (4).
JP Z,adres; sprong naar regel 9 (5).
INC HL; verhoog de pointer (6).
DEC BC; verlaag de teller (7).
JP NZ,adres; sprong naar regel 4 (8).
Volgende instructie (9).

Het beginadres van het geheugenblok wordt als pointer in HL geladen (regel 1). De teller moet bij de aanvang van het programma de grootte van het geheugenblok bevatten (regel 2). In regel 3 wordt de Accu van de te vergelijken data voorzien. In regel 4 wordt deze vergeleken met de data in het register dat door de pointer wordt aangewezen. Bij gelijke data wordt de Z- FLAG gereset, zodat door regel 5 een sprong naar het einde van het programma wordt gemaakt. Het adres van het desbetreffende geheugenregister wordt gevonden door HL uit te lezen. Is er geen gelijke data gevonden, dan vervolgt het programma met het verhogen van HL en het verlagen van de teller. Is deze nog niet nul, dan is het gehele blok nog niet onderzocht en wordt teruggegaan naar regel 4 voor de volgende geheugenplaats. De regels 4, 6 en 7 worden uitgevoerd door de instructie

CPI

ComPare and Increment.
Operatiecode: 11101101 10100001 (EDA1H).

De instructievolgorde die met CPI kan worden toegepast is:

LD HL,E100 (1).
LD BC,0100 (2).
LD A,A3 (3).
CPI (4).
JP Z,adres; sprong naar regel 7 (5).
JP PE,adres; sprong naar regel 4 (6).
Volgende instructie (7).

Zodra de vergelijking in regel 4 resultaat heeft opgeleverd wordt in regel 5 naar het einde van het programma gesprongen. Het gezochte adres is dan weer uit HL te lezen. Door het verhogen van HL na de vergelijking geeft deze echter één adres te hoog aan. Heeft de vergelijking geen resultaat gehad, dan wordt in regel 4 nagegaan of alle geheugenplaatsen zijn behandeld. Bij het nul worden van BC wordt door CPI de V/P-FLAG gereset, anders geset. Dit komt overeen met de werking van de LDI-instructie. De regels 4 tot en met 6 van dit laatste programma worden gecombineerd door

CPIR

ComPare, Increment and Repeat.
Operatiecode: 11101101 10110001 (ED81H).
De instructievolgorde wordt nu:

LD HL,E100
LD BC,0100
LD A,A3
CPIR
Volgende instructie.

Bij CPI en CPIR wordt het geheugenblok van laag naar hoog doorlopen. Andersom kan het ook:

CPD

ComPare and Delete.
Operatiecode: 11101101 10101001 (EDA9H).

CPDR

ComPare, Delete and Repeat.
Operatiecode: 11101101 10111001 (EDB9H).

De instructievolgorde wordt:

LD HL,E1FF
LD BC,0100
LD A,A3
CPDR
Volgende instructie.

De blokvergelijkingsinstructies zijn samengevat in lijst 14.

SEARCH LOCATION	
REG. INDIR.	
(HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

Lijst 14. De blokvergelijkingsinstructies.

6.11. Instructies voor subroutines.

Evenals bij de BASIC programmataal kennen we bij machinetaalprogramma's ook het begrip "subroutine". De subroutine is een programmadeel dat binnen een bepaald programma meerdere keren wordt gebruikt. Het desbetreffende programma, het "hoofdprogramma", wordt onderbroken om een sprong te maken naar een ander programmadeel, de subroutine. Na het doorlopen van de subroutine wordt het hoofdprogramma weer voortgezet waar het voor de sprong naar de subroutine was verlaten. Dat betekent dat het adres van het hoofdprogramma waar weer naar teruggesprongen moet worden, door de processor moet worden bewaard. Hiervoor wordt de STACK gebruikt. Het springen naar een subroutine wordt het *"aanroepen"* genoemd en de instructie die de sprong tewerkstelt is dan ook een CALL-instructie. Een CALL-instructie heeft tot resultaat dat eerst de inhoud van de programmateller (PC) in de volgorde lage byte - hoge byte naar de STACK wordt geschreven. In figuur 41 is een dergelijke situatie geschetst. Daarna wordt de programmateller voorzien van het adres van de subroutine. De processor vervolgt daardoor zijn taak met het afwerken van de instructies van de subroutine. Aan het eind van de subroutine dient een instructie te komen die de processor laat weten dat hij weer terug moet naar het hoofdprogramma. Dit is de instructie RET van "RETurn". Zodra deze instructie door de processor wordt ontvangen ,wordt het bewaarde adres uit de STACK gelezen en geplaatst in de programmateller. Geheel overeenkomstig de werking van de STACK geschiedt dat nu in de volgorde hoge byte - lage byte. Het springen naar een subroutine betekent in werkelijkheid niet dat het hoofdprogramma wordt onderbroken zoals dat bij een interrupt gebeurt. De plaats waar de subroutine wordt aangeroepen ligt in het programma vast en is niet willekeurig, wat bij een interrupt wel het geval is. Wel moet er rekening mee worden gehouden dat in de subroutine een aantal processorregisters zullen worden gebruikt waardoor de oorspronkelijke inhoud van deze registers verloren kan gaan. Om dit te voorkomen kunnen twee oplossingen worden gebruikt. Ten eerste kunnen aan het begin van de subroutine de inhouden van de registers die in de routine worden gebruikt, naar de STACK worden geschreven, zoals dat in paragraaf 6.2. is beschreven bij de STACK-handelingen voor de interruptroutine. Aan het eind van de subroutine kunnen de registers hun oorspronkelijke inhoud weer terugkrijgen. Dit is de makkelijkste methode omdat in het hoofdprogramma geen rekening hoeft te worden gehouden met het eventueel verloren gaan van gegevens. Een tweede oplossing is om de inhouden van de registers naar de STACK te schrijven voor het aanroepen van de subroutine en deze weer uit te lezen na het vervolgen van het hoofdprogramma. Het voordeel hiervan is dat niet meer gegevens naar de STACK worden geschreven dan strikt noodzakelijk is. Worden *binnen* een subroutine STACK-handelingen verricht, dan dient er steeds voor te worden gezorgd dat aan het eind van de routine de Stackpointer dezelfde waarde heeft als aan het begin (bijvoorbeeld: evenveel POP-instructies als PUSH-instructies). Dat is nodig om de program-

mateller met het goede adres te kunnen laden voor het terugkeren naar het hoofdprogramma.
Verder is het gebruik van subroutines gelijk aan dat bij BASIC programma's. Ook kan binnen een subroutine opnieuw een subroutine worden aangeroepen.

CALL nn

Deze instructie heeft een onvoorwaardelijke sprong naar een subroutine tot gevolg waarvan het adres wordt gevormd door de twee bytes nn, direct volgend op de operatiecode:

76	543	210	bitnummer.
11	001	101	byte1.
n.	...	...	byte2.
n.	...	...	byte3.

CALL cc,nn

Deze instructie veroorzaakt een sprong naar een subroutine op het adres dat wordt gevormd door de bytes nn en onder de voorwaarde cc. De sprong wordt niet gemaakt als niet aan de voorwaarde is voldaan.

Operatiecode:

76	543	210	bitnummer.
11	cc.	100	byte1.
n.	...	...	byte2.
n.	...	...	byte3.

Voor cc dienen drie bits te worden ingevuld die gevonden kunnen worden in de volgende tabel:

cc. Voorwaarde.
000 NZ, Non Zero
001 Z, Zero
010 NC, Non Carry
011 C, Carry
100 PO, Parity Odd
101 PE, Parity Even
110 P, sign Positive
111 M, sign Minus

RET

Deze instructie veroorzaakt een onvoorwaardelijke sprong terug naar het onderbroken programma.

Operatiecode: 11001001 (C9H).

RET cc

Het teruggaan naar het onderbroken programma is hierbij afhankelijk van de voorwaarde cc. Wordt niet aan de voorwaarde voldaan, dan wordt het programma van de subroutine voortgezet.

Operatiecode:

76	543	210	bitnummer.
11	cc.	000	byte1.

Voor cc dienen drie bits te worden ingevuld die kunnen worden gevonden in de voorgaande tabel.

Het principe van de interruptroutine (hoofdstuk 5) verschilt niet veel van die van de subroutine, alleen de methode van aanroepen is verschillend. Het weer teruggaan naar het onderbroken programma vraagt echter een andere instructie dan die aan het einde van een subroutine:

RETI

Deze instructie wordt gebruikt als afsluiting van een interruptroutine die is aangeroepen door een INT (maskeerbaar).

Operatiecode: 11101101 01001101 (ED4DH).

De niet-maskeerbare interrupt (NMI) vraagt een andere instructie om terug te keren naar het onderbroken programma. Aan het eind van een NMI-interrupt moet de inhoud van IFF2 verplaatst worden naar IFF1. Daarna kan de programmateller weer van het juiste adres worden voorzien. De instructie is:

RETN

Operatiecode: 11101101 01000101 (ED45H).

De instructies voor de subroutine en interruptroutine behandeling zijn samengevat in lijst 12.

6.12. Restartinstructies.
De restartinstructies (RST) kunnen worden vergeleken met de CALL- instructies. Er kunnen echter slechts subroutines mee worden aangeroepen die

op specifieke adressen aanvangen van pagina 0 van de geheugenruimte. Omdat slechts van pagina 0 gebruik wordt gemaakt spreekt men wel van "(beperkte) Zero Page Addressing mode". Het Mnemonic symbool is

RST p

Hierin is p een getal dat de geheugenplaats op pagina 0 aangeeft waar de desbetreffende subroutine aanvangt. Voor p kan gekozen worden uit 00H, 08H, 10H, 18H, 20H, 28H, 30H of 38H. Wilt u een subroutine op het adres 0020H aanroepen, dan gebruikt u

RST 20H.

Operatiecode:

76	543	210	bitnummer.
11	t..	111	byte1.

Voor t dienen drie bits te worden ingevuld die gevonden kunnen worden in onderstaande tabel:

t..	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Aan het eind van de desbetreffende subroutine moet weer een RET- instructie worden geplaatst.

De routine die op geheugenplaats 0000H aanvangt is een bijzondere routine. Na het inschakelen van de computer zal de processor beginnen met een aantal van zijn interne registers een bepaalde waarde te geven. Hierbij wordt de programmateller 0 en dat getal wordt dan ook op de databus geplaatst. Deze geeft daarom direct na het inschakelen het adres 0000H aan. Als laatste handeling van deze *"initiatie"* wordt aan de besturingslijnen een zodanige waarde gegeven dat de data op het adres 0000H kan worden gelezen. Deze data is de eerste operatiecode van een programma dat de computer "in de starthouding" moet brengen. Vaak houdt dat in dat het geheugen wordt getest en dat aan diverse geheugenplaatsen een bepaalde waarde wordt gegeven. In elk geval moet hierdoor ook het systeemprogramma worden gestart, zodat de computer gaat wachten op het invoeren van gegevens vanaf het toetsenbord of vanuit een ander randapparaat. Dit geheel, vanaf het inschakelen van de netspanning, wordt de *"koude start"* van de computer genoemd. Het starten van het systeemprogramma op het adres 0000H is de *"warme start"*. De netspanning is dan al ingeschakeld en de processor is in de situatie dat hij zijn werk kan beginnen. Verwerkt u de instructie RST 00H in een programma, dan veroorzaakt deze een warme start. Dat betekent dat uw computer daarna weer helemaal "blank" is, zoals direct na het inschakelen. Alle gegevens en het programma zijn uit het geheugen verdwenen.

De restartinstructies zijn samengevat in lijst 15.

		OP CODE	
CALL ADDRESS	0000_H	C7	'RST 0'
	0008_H	CF	'RST 8'
	0010_H	D7	'RST 16'
	0018_H	DF	'RST 24'
	0020_H	E7	'RST 32'
	0028_H	EF	'RST 40'
	0030_H	F7	'RST 48'
	0038_H	FF	'RST 56'

Lijst 15. Restartinstructies.

6.13. In- en uitvoerinstructies.

Met in- en uitvoerinstructies worden de instructies bedoeld die betrekking hebben op het lezen van of het schrijven naar respectievelijk de ingangspoorten of de uitgangspoorten. Ze staan beter bekend als input respectievelijk output instructies en we zullen deze uitdrukkingen dan ook verder gebruiken. Acht poorten zijn samengevoegd tot een re-

gister en zij vormen de verbinding tussen de computer en de buitenwereld (het randapparaat).

Zowel de ingangs- als de uitgangspoorten kunnen slechts door 8 adreslijnen worden geadresseerd (paragraaf 1.1.). In het algemeen zullen hiervoor de adreslijnen A0 tot en met A7 worden gebruikt. Het is ook mogelijk om in plaats daarvan de adreslijnen A8 tot en met A15 te gebruiken. Hiermee is rekening gehouden bij de instructies. Het is hiermee zowel mogelijk de acht lage adreslijnen van een getal te voorzien als de acht hoge. Dat betekent dat een instructie zowel de lage als de hoge adreslijnen een inhoud geeft. Een input- of een outputinstructie zal tot gevolg hebben dat via de besturingslijnen slechts een ingangs- of een uitgangspoort wordt geactiveerd. Een geheugenelement dat zich op het adres bevindt dat bij een in/out bewerking op de adreslijnen staat, zal niet actief worden.
Er zijn poorten die slechts ingang kunnen zijn en andere die slechts uitgang kunnen zijn. Er zijn echter ook schakelingen die zowel als ingangspoort of als uitgangspoort geschakeld kunnen worden. In dat geval moeten de poorten eerst tot ingang of tot uitgang worden gemaakt door een codegetal in een speciaal register te plaatsen. Het adres van dit register en de samenstelling van het codegetal moeten dan blijken uit de gebruiksaanwijzing van de computer.

IN A,(n)

Operatiecode:

76	543	210	bitnummer.
11	011	011	byte1.
n.	...	...	byte2.

Deze instructie heeft *geen* invloed op de FLAGS. Bij deze inputinstructie wordt het adres gevormd door de adresbuffers A0 tot en met A7 te laden met de waarde van byte2. De adresbuffers A8 tot en met A15 worden voorzien van de inhoud van de Accu. Moet de hoge byte van het adres een bepaalde waarde hebben, dan dient dus vooraf de Accu van de juiste inhoud te worden voorzien. Worden alleen de lage adreslijnen gebruikt voor het adresseren van het poortregister, dan kan de waarde van de Accu bij de aanvang van de instructie willekeurig zijn. Aan het eind van de instructie heeft de Accu dezelfde waarde als het poort-register. Een poortregister is opgebouwd uit acht poorten. Vaak is het nodig dat de waarde van slechts een enkele poort uit het register moet worden gemeten. In dat geval moet de instructie IN A,(n) worden opgevolgd door BIT b,A. Deze laatste instructie geeft de mogelijkheid de waarde van de poort (door middel van de Z-FLAG) vast te stellen.

IN r,(C)

Operatiecode:

76	543	210	bitnummer.
11	101	101	byte1
01	r..	000	byte2.

Deze instructie heeft invloed op de FLAGS.
Bij deze instructie vormt de inhoud van het registerpaar BC het adres van het desbetreffende poortregister. Zijn hiervoor slechts de adreslijnen A0 tot en met A7 nodig, dan is de inhoud van register B voor het adres van geen waarde. Na het afwerken van de instructie heeft het register r de waarde van het poortregister. Voor r in de instructie dienen de drie bits ingevuld te worden die afhankelijk zijn van het gekozen register.

r..	register.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

In bovenstaande tabel komt de bitcombinatie 110 niet voor. Wordt deze combinatie voor r in de instructiecode gevoerd, dan wordt wel de poort gelezen maar het resultaat niet in een register opgeslagen. De desbetreffende FLAGS zijn echter, afhankelijk van het resultaat, geset of gereset.

OUT (n),A

Operatiecode:

76	543	210	bitnummer.
11	010	011	byte1.
n.	...	...	byte2.

Ook bij deze *outputinstructie* wordt het adres gevormd door de inhoud van n in de adresbuffers A0 tot en met A7 te plaatsen en de inhoud van de Accu in de adresbuffers A8 tot en met A15 te schrijven. Dit laatste kan echter niet van waarde zijn. De Accu moet het getal bevatten dat naar het outputregister moet worden geschreven en het zou wel toevallig zijn als dit gelijk was aan de hoge byte van het adres.

OUT (C),r

Nu wordt het adres weer gevormd door de inhoud van het registerpaar BC. Is alleen de lage byte nodig om te adresseren dan is de inhoud van register B niet van belang. De inhoud van het desbetreffende register r wordt naar het poortregister geschreven.
Het kan gebeuren dat via eenzelfde poortregister een reeks van data uit een randapparaat moet worden ingevoerd. Het poortregister moet dan steeds na elkaar worden gelezen terwijl na elke keer lezen van de poorten de data in een ander geheugenelement moet worden opgeslagen. Zo wordt dan een blok van het datageheugen gevuld. Dit geheugenblok kan ook een *"buffer"* worden genoemd. Om dit met zo weinig mogelijk instructies mogelijk te maken zijn er de *blokinvoerinstructies*.

INI

Operatiecode: 11101101 10100010 (EDA2H).

Hierbij is alleen adresseren met de adreslijnen A0 tot en met A7 mogelijk. De inhoud van register C is onveranderlijk en geeft het adres van de poort. De inhoud van register B fungeert als teller. Het grootste aantal keren dat de ingangspoort achter elkaar kan worden gelezen is daardoor 256. Het registerpaar HL bevat het adres van het dataregister waarin de data moet worden opgeslagen. De instructie is een samenvatting van de volgende opdrachten:

IN A,(C); lees de ingangspoort.
LD (HL),A; schrijf data in het geheugen.
INC HL; verhoog adres van het geheugenregister.
DEC B; verlaag de teller.

De instructie vult een geheugenblok van laag naar hoog. Voor het toepassen van de instructie moet eerst HL met het aanvangsadres van het geheugenblok, C met het adres van het poortregister en B met het aantal te lezen bytes worden geladen. De instructie leest de poort slechts één keer en moet steeds worden herhaald tot de teller op nul staat. De volgende instructie heeft dezelfde werking maar herhaalt zichzelf tot B nul is:

INIR

Operatiecode: 11101101 10110010 (EDB2H).

Een geheugenblok kan ook van hoog naar laag worden gevuld met de data:

IND

Operatiecode: 11101101 10101010 (EDAAH).

De instructie heeft dezelfde werking als INI. Nu wordt echter niet de inhoud van HL met 1 vermeerderd maar met 1 verminderd. Dezelfde betrekking zien we tussen INIR en

INDR

Operatiecode: 11101101 10111010 (EDBAH).

Op dezelfde manier als een blok data kan worden ingelezen, kan ook een blok data worden uitgestuurd naar het randapparaat. Bij de instructie

OUTI

Operatiecode: 11101101 10100011 (EDA3H).

bevat C het adres van de uitgangspoort waarin de data moet worden geschreven, HL het adres van het dataregister dat moet worden gelezen en is B weer de teller. Hierbij blijft C constant (steeds dezelfde poort wordt ingeschreven), wordt HL met 1 opgehoogd en wordt B met 1 verlaagd. Het voortdurend herhalen van deze instructie heeft tot gevolg dat een blok van het datageheugen byte voor byte wordt uitgelezen (van laag naar hoog) en dat de data hieruit na elkaar in het desbetreffende poortregister wordt geschreven.

OTIR

Operatiecode: 11101101 10110011 (EDB3H).

INPUT DESTINATION				PORT ADDRESS: IMMED. n	PORT ADDRESS: REG. INDIR. (C)	
	INPUT 'IN'	REG ADDRESSING	A	DB	ED 78	
	INPUT 'IN'	REG ADDRESSING	B		ED 40	
	INPUT 'IN'	REG ADDRESSING	C		ED 48	
	INPUT 'IN'	REG ADDRESSING	D		ED 50	
	INPUT 'IN'	REG ADDRESSING	E		ED 58	
	INPUT 'IN'	REG ADDRESSING	H		ED 60	
	INPUT 'IN'	REG ADDRESSING	L		ED 68	
	'INI' – INPUT & Inc HL, Dec B	REG. INDIR	(HL)		ED A2	BLOCK INPUT COMMANDS
	'INIR' – INP, Inc HL, Dec B, REPEAT IF B≠0	REG. INDIR	(HL)		ED B2	BLOCK INPUT COMMANDS
	'IND' – INPUT & Dec HL, Dec B	REG. INDIR	(HL)		ED AA	BLOCK INPUT COMMANDS
	'INDR' – INPUT, Dec HL, Dec B, REPEAT IF B≠0	REG. INDIR	(HL)		ED BA	BLOCK INPUT COMMANDS

Lijst 16. De invoerinstructies.

Deze instructie heeft dezelfde werking maar herhaalt die steeds tot de teller B op 0 staat.

Het geheugenblok kan ook van hoog naar laag worden uitgelezen:

OUTD

Operatiecode: 11101101 10101011 (EDABH).

OTDR

Operatiecode: 11101101 10111011 (EDBBH).

Deze laatste instructie herhaalt de werking tot de teller B 0 is.
De invoerinstructies zijn samengevat in lijst 16 en de uitvoerinstructies in lijst 17.

6.14. Diversen.
Een aantal instructies is niet direct in een bepaalde groep onder te brengen, vandaar de groep ''diversen''.

NOP

No OPeration.

Operatiecode: 00000000 (00H).

Het kan voorkomen dat een programma moet worden gewijzigd en dat er daardoor ergens een paar instructies zijn verdwenen. Een programma is echter een aaneengesloten reeks van instructies zodat lege geheugenplaatsen in een programma niet mogen voorkomen. Om nu te voorkomen dat gehele blokken moeten worden opgeschoven (en eventuele sprongadressen opnieuw moeten worden ingevoerd) kunnen de lege geheugenplaatsen worden opgevuld met de instructie NOP. Deze instructie heeft geen enkele werking en de inhoud van de registers verandert daardoor niet. Ook kan NOP worden toegepast in programma's waarin een vertraging moet worden aangebracht. Het niet uitvoeren van een werking kost de processor dus ook enige tijd (vier kloktijden)!

Lijst 17. De uitvoerinstructies.

			SOURCE								
			REGISTER							REG. IND.	
			A	B	C	D	E	H	L	(HL)	
'OUT'	IMMED.	n	D3 n								
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69		
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3	BLOCK OUTPUT COMMANDS
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3	
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB	
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB	
	PORT DESTINATION ADDRESS										

HALT

Operatiecode: 01110110 (76H).

Als de processor deze instructie tegenkomt, dan stopt hij met het afwerken van het programma. Eventuele volgende instructies worden niet gelezen. De processor gaat over in het uitvoeren van de instructie NOP en herhaalt dat tot een interrupt hier een einde aan maakt. Een interrupt is (behalve het uitschakelen van de computer) dus het enige waarop de processor nog reageert.

DI

Disable interrupts.
Operatiecode: 11110011 (F3H).

Met deze instructie kan worden voorkomen dat de processor reageert op het ''aanvragen'' van een INT interrupt (paragraaf 5.4.). Deze instructie reset de interrupt flip flop IFF1.

EI

Enable interrupt.
Operatiecode: 11111011 (FBH).

Door deze instructie wordt IFF1 geset, zodat de processor kan reageren op een INT interruptaanvraag.

IM 0

Set interruptmode 0.
Operatiecode: 11101101 01000110 (ED46H).

IM 1

Set interruptmode 1.
Operatiecode: 11101101 01010110 (ED56H).

IM 2

Set interruptmode 2.
Operatiecode: 11101101 01011110 (ED5EH).

De instructies in deze paragraaf zijn samengevat in lijst 18.

Lijst 18. Diversen.

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

Instructieset
Z 80

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, s	r ← s	•	•	X	•	X	•	•	•	01 r s		1	1	4	r, s Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000 B
										← n →					001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	011 E
										01 r 110					100 H
										← d →					101 L
LD r, (IY+d)	r ← (IY+d)	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	111 A
										01 r 110					
										← d →					
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	
										01 110 r					
										← d →					
LD (IY+d), r	(IY+d) ← r	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	
										01 110 r					
										← d →					
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110	36	2	3	10	
										← n →					
LD (IX+d), n	(IX+d) ← n	•	•	X	•	X	•	•	•	11 011 101	DD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD (IY+d), n	(IY+d) ← n	•	•	X	•	X	•	•	•	11 111 101	FD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010	3A	3	4	13	
										← n →					
										← n →					
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010	32	3	4	13	
										← n →					
										← n →					
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 010 111	57				
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 011 111	5F				
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 000 111	47				
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 001 111	4F				

Lijst 19. De instructieset.

Mnemonic	Symbolic Operation	Flags								Op-Code		No. of Bytes	No. of M Cycles	No. of T States	Comments
		S	Z		H		P/V	N	C	76 543 210	Hex				
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001		3	3	10	dd Pair
										← n →					00 BC
										← n →					01 DE
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101	DD	4	4	14	10 HL
										00 100 001	21				11 SP
										← n →					
										← n →					
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101	FD	4	4	14	
										00 100 001	21				
										← n →					
										← n →					
LD HL, (nn)	H ← (nn+1)	•	•	X	•	X	•	•	•	00 101 010	2A	3	5	16	
	L ← (nn)									← n →					
										← n →					
LD dd, (nn)	dd_H ← (nn+1)	•	•	X	•	X	•	•	•	11 101 101	ED	4	6	20	
	dd_L ← (nn)									01 dd1 011					
										← n →					
										← n →					
LD IX, (nn)	IX_H ← (nn+1)	•	•	X	•	X	•	•	•	11 011 101	DD	4	6	20	
	IX_L ← (nn)									00 101 010	2A				
										← n →					
										← n →					
LD IY, (nn)	IY_H ← (nn+1)	•	•	X	•	X	•	•	•	11 111 101	FD	4	6	20	
	IY_L ← (nn)									00 101 010	2A				
										← n →					
										← n →					
LD (nn), HL	(nn+1) ← H	•	•	X	•	X	•	•	•	00 100 010	22	3	5	16	
	(nn) ← L									← n →					
										← n →					
LD (nn), dd	(nn+1) ← dd_H	•	•	X	•	X	•	•	•	11 101 101	ED	4	6	20	
	(nn) ← dd_L									01 dd0 011					
										← n →					
										← n →					
LD (nn), IX	(nn+1) ← IX_H	•	•	X	•	X	•	•	•	11 011 101	DD	4	6	20	
	(nn) ← IX_L									00 100 010	22				
										← n →					
										← n →					
LD (nn), IY	(nn+1) ← IY_H	•	•	X	•	X	•	•	•	11 111 101	FD	4	6	20	
	(nn) ← IY_L									00 100 010	22				
										← n →					
										← n →					
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10	
										11 111 001	F9				
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10	
										11 111 001	F9				qq Pair
PUSH qq	(SP-2) ← qq_L	•	•	X	•	X	•	•	•	11 qq0 101		1	3	11	00 BC
	(SP-1) ← qq_H														01 DE
PUSH IX	(SP-2) ← IX_L	•	•	X	•	X	•	•	•	11 011 101	DD	2	4	15	10 HL
	(SP-1) ← IX_H									11 100 101	E5				11 AF
PUSH IY	(SP-2) ← IY_L	•	•	X	•	X	•	•	•	11 111 101	FD	2	4	15	
	(SP-1) ← IY_H									11 100 101	E5				
POP qq	qq_H ← (SP+1)	•	•	X	•	X	•	•	•	11 qq0 001		1	3	10	
	qq_L ← (SP)														
POP IX	IX_H ← (SP+1)	•	•	X	•	X	•	•	•	11 011 101	DD	2	4	14	
	IX_L ← (SP)									11 100 001	E1				
POP IY	IY_H ← (SP+1)	•	•	X	•	X	•	•	•	11 111 101	FD	2	4	14	
	IY_L ← (SP)									11 100 001	E1				

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE↔HL	•	•	X	•	X	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF↔AF'	•	•	X	•	X	•	•	•	00 001 000	08	1	1	4	
EXX	(BC↔BC' DE↔DE' HL↔HL')	•	•	X	•	X	•	•	•	11 011 001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔(SP+1) L ↔(SP)	•	•	X	•	X	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H↔(SP+1) IX_L↔(SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H↔(SP+1) IY_L↔(SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 100 000	ED A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A – (HL) HL ← HL+1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A – (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A – (HL) HL ← HL-1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A – (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	X	↕	X	V	0	↕	10 [000] r		1	1	4	r Reg.
ADD A, n	A ← A + n	↕	↕	X	↕	X	V	0	↕	11 [000] 110		2	2	7	000 B
										← n →					001 C
															010 D
ADD A, (HL)	A ← A+(HL)	↕	↕	X	↕	X	V	0	↕	10 [000] 110		1	2	7	011 E
ADD A, (IX+d)	A ← A+(IX+d)	↕	↕	X	↕	X	V	0	↕	11 011 101	DD	3	5	19	100 H
										10 [000] 110					101 L
										← d →					111 A
ADD A, (IY+d)	A ← A+(IY+d)	↕	↕	X	↕	X	V	0	↕	11 111 101	FD	3	5	19	
										10 [000] 110					
										← d →					
ADC A, s	A ← A+s+CY	↕	↕	X	↕	X	V	0	↕	[001]					s is any of r, n,
SUB s	A ← A - s	↕	↕	X	↕	X	V	1	↕	[010]					(HL), (IX+d),
SBC A, s	A ← A - s - CY	↕	↕	X	↕	X	V	1	↕	[011]					(IY+d) as shown for
AND s	A ← A ∧ s	↕	↕	X	1	X	P	0	0	[100]					ADD instruction.
OR s	A ← A ∨ s	↕	↕	X	0	X	P	0	0	[110]					The indicated bits
XOR s	A ← A ⊕ s	↕	↕	X	0	X	P	0	0	[101]					replace the [000] in
CP s	A - s	↕	↕	X	↕	X	V	1	↕	[111]					the ADD set above.
INC r	r ← r + 1	↕	↕	X	↕	X	V	0	•	00 r [100]		1	1	4	
INC (HL)	(HL) ← (HL)+1	↕	↕	X	↕	X	V	0	•	00 110 [100]		1	3	11	
INC (IX+d)	(IX+d) ←	↕	↕	X	↕	X	V	0	•	11 011 101	DD	3	6	23	
	(IX+d)+1									00 110 [100]					
										← d →					
INC (IY+d)	(IY+d) ←	↕	↕	X	↕	X	V	0	•	11 111 101	FD	3	6	23	
	(IY+d)+1									00 110 [100]					
										← d →					
DEC s	s ← s - 1	↕	↕	X	↕	X	V	1	•	[101]					s is any of r, (HL),
															(IX+d), (IY+d) as
															shown for INC.
															DEC same format
															and states as INC.
															Replace [100] with
															[101] in OP Code.

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc, content into packed BCD following add or subtract with packed BCD operands	↕	↕	X	↕	X	P	•	↕	00 100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	X	1	X	•	1	•	00 101 111	2F	1	1	4	Complement accumulator (One's complement)
NEG	$A \leftarrow \overline{A}+1$	↕	↕	X	↕	X	V	1	↕	11 101 101 01 000 100	ED 44	2	2	8	Negate acc, (two's complement)
CCF	$CY \leftarrow \overline{CY}$	•	•	X	X	X	•	0	↕	00 111 111	3F	1	1	4	Complement carry flag
SCF	$CY \leftarrow 1$	•	•	X	0	X	•	0	1	00 110 111	37	1	1	4	Set carry flag
NOP	No operation	•	•	X	•	X	•	•	•	00 000 000	00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110	76	1	1	4	
DI*	$IFF \leftarrow 0$	•	•	X	•	X	•	•	•	11 110 011	F3	1	1	4	
EI*	$IFF \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 011	FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 01 000 110	ED 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 01 010 110	ED 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 01 011 110	ED 5E	2	2	8	

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments	
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	↕	00 ss1 001		1	3	11	ss	Reg.
															00	BC
ADC HL, ss	HL ← HL+ss+CY	↕	↕	X	X	X	V	0	↕	11 101 101	ED	2	4	15	01	DE
										01 ss1 010					10	HL
															11	SP
SBC HL, ss	HL ← HL-ss-CY	↕	↕	X	X	X	V	1	↕	11 101 101	ED	2	4	15		
										01 ss0 010						
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101	DD	2	4	15	pp	Reg.
										00 pp1 001					00	BC
															01	DE
															10	IX
															11	SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101	FD	2	4	15	rr	Reg.
										00 rr1 001					00	BC
															01	DE
															10	IY
															11	SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6		
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10		
										00 100 011	23					
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10		
										00 100 011	23					
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6		
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10		
										00 101 011	2B					
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10		
										00 101 011	2B					

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
RLCA	CY ← 7 ← 0 ← A	•	•	X	0	X	•	0	↕	00 000 111	07	1	1	4	Rotate left circular accumulator
RLA	CY ← 7 ← 0 ← A	•	•	X	0	X	•	0	↕	00 010 111	17	1	1	4	Rotate left accumulator
RRCA	→ 7 → 0 → CY A	•	•	X	0	X	•	0	↕	00 001 111	0F	1	1	4	Rotate right circular accumulator
RRA	→ 7 → 0 → CY A	•	•	X	0	X	•	0	↕	00 011 111	1F	1	1	4	Rotate right accumulator
RLC r		↕	↕	X	0	X	P	0	↕	11 001 011 00 000 r	CB	2	2	8	Rotate left circula. register r
RLC (HL)		↕	↕	X	0	X	P	0	↕	11 001 011 00 000 110	CB	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)	CY ← 7 ← 0 ← r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	11 011 101 11 001 011 ← d → 00 000 110	DD CB	4	6	23	
RLC (IY+d)		↕	↕	X	0	X	P	0	↕	11 111 101 11 001 011 ← d → 00 000 110	FD CB	4	6	23	
RL s	CY ← 7 ← 0 ← s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	010					Instruction format and states are as shown for RLC's. To form new Op-Code replace 000 of RLC's with shown code
RRC s	→ 7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	001					
RR s	→ 7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	011					
SLA s	CY ← 7 ← 0 ← 0 s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	100					
SRA s	7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	101					
SRL s	0 → 7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	↕	↕	X	0	X	P	0	↕	111					
RLD	A 7-4 3-0 7-4 3-0 (HL)	↕	↕	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	Rotate digit left and right between the accumulator and location (HL).
RRD	A 7-4 3-0 7-4 3-0 (HL)	↕	↕	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	The content of the upper half of the accumulator is unaffected

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments	
BIT b, r	$Z \leftarrow \overline{r}_b$	X	↕	X	1	X	X	0	•	11 001 011	CB	2	2	8	r	Reg.
										01 b r					000	B
BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	X	↕	X	1	X	X	0	•	11 001 011	CB	2	3	12	001	C
										01 b 110					010	D
BIT b, $(IX+d)_b$	$Z \leftarrow \overline{(IX+d)}_b$	X	↕	X	1	X	X	0	•	11 011 101	DD	4	5	20	011	E
										11 001 011	CB				100	H
										← d →					101	L
										01 b 110					111	A
															b	Bit Tested
BIT b, $(IY+d)_b$	$Z \leftarrow \overline{(IY+d)}_b$	X	↕	X	1	X	X	0	•	11 111 101	FD	4	5	20	000	0
										11 001 011	CB				001	1
										← d →					010	2
										01 b 110					011	3
															100	4
															101	5
															110	6
															111	7
SET b, r	$r_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011	CB	2	2	8		
										[11] b r						
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011	CB	2	4	15		
										[11] b 110						
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 011 101	DD	4	6	23		
										11 001 011	CB					
										← d →						
										[11] b 110						
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 101	FD	4	6	23		
										11 001 011	CB					
										← d →						
										[11] b 110						
RES b, s	$s_b \leftarrow 0$	•	•	X	•	X	•	•	•	[10]					To form new Op-Code replace [11] of SET b, s with [10]. Flags and time states for SET instruction	
	$s \equiv r$, (HL), (IX+d), (IY+d)															

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011	C3	3	3	10	
										← n →					
										← n →					cc Condition
JP cc, nn	If condition cc	•	•	X	•	X	•	•	•	11 cc 010		3	3	10	000 NZ non zero
	is true PC ← nn,									← n →					001 Z zero
	otherwise									← n →					010 NC non carry
	continue														011 C carry
															100 PO parity odd
															101 PE parity even
															110 P sign positive
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	00 011 000	18	2	3	12	111 M sign negative
										← e-2 →					
JR C, e	If C = 0,	•	•	X	•	X	•	•	•	00 111 000	38	2	2	7	If condition not met
	continue									← e-2 →					
	If C = 1,											2	3	12	If condition is met
	PC ← PC+e														
JR NC, e	If C = 1,	•	•	X	•	X	•	•	•	00 110 000	30	2	2	7	If condition not met
	continue									← e-2 →					
	If C = 0,											2	3	12	If condition is met
	PC ← PC+e														
JR Z, e	If Z = 0	•	•	X	•	X	•	•	•	00 101 000	28	2	2	7	If condition not met
	continue									← e-2 →					
	If Z = 1,											2	3	12	If condition is met
	PC ← PC+e														
JR NZ, e	If Z = 1,	•	•	X	•	X	•	•	•	00 100 000	20	2	2	7	If condition not met
	continue									← e-2 →					
	If Z = 0,											2	3	12	If condition is met
	PC ← PC+e														
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	8	
										11 101 001	E9				
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	8	
										11 101 001	E9				
DJNZ, e	B ← B-1	•	•	X	•	X	•	•	•	00 010 000	10	2	2	8	If B = 0
	If B = 0,									← e-2 →					
	continue														
	If B ≠ 0,											2	3	13	If B ≠ 0
	PC ← PC+e														

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	(SP-1) → PC_H	•	•	X	•	X	•	•	•	11 001 101	CD	3	5	17	
	(SP-2) → PC_L									← n →					
	PC ← nn									← n →					
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	X	•	X	•	•	•	11 cc 100		3	3	10	If cc is false
										← n →					
										← n →		3	5	17	If cc is true
RET	PC_L ← (SP)	•	•	X	•	X	•	•	•	11 001 001	C9	1	3	10	
	PC_H ← (SP+1)														
RET cc	If condition cc is false	•	•	X	•	X	•	•	•	11 cc 000		1	1	5	If cc is false
	continue, otherwise same as RET											1	3	11	If cc is true
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 101 101	ED	2	4	14	
										01 001 101	4D				
RETN[1]	Return from non maskable interrupt	•	•	X	•	X	•	•	•	11 101 101	ED	2	4	14	
										01 000 101	45				
RST p	(SP-1) → PC_H	•	•	X	•	X	•	•	•	11 t 111		1	3	11	
	(SP-2) → PC_L														
	PC_H ← 0														
	PC_L ← p														

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Lijst 19. De instructieset (vervolg).

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 ← n →	DB	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 01 r 000	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 100 010	ED A2	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 110 010	ED B2	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 101 010	ED AA	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 111 010	ED BA	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11 010 011	D3	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 101 101 01 r 001	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	B ← B - 1 (C) ← (HL) HL ← HL + 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 100 011	ED A3	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	B ← B - 1 (C) ← (HL) HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 110 011	ED B3	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 101 011	ED AB	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 111 011	ED BB	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

Lijst 19. De instructieset (vervolg).

DEEL 2. DE MSX-COMPUTER

7. Inleiding

7.1. De indeling van de geheugenruimte.
Om te weten op welke plaats in de geheugenruimte een machineprogramma kan worden geschreven is het nodig om de indeling van de geheugenruimte van de MSX-computer te kennen. Om te beginnen de geheugenindeling zoals die zich aan ons voordoet bij het inschakelen van de computer. Zoals u nu bekend is kan de Z80 processor een geheugenomvang van 64 Kbyte adresseren. Deze geheugenruimte is verdeeld in blokken van 16 Kbyte die elk een *"page"* worden genoemd. Om geen verwarring te krijgen met de "normale" indeling in pagina's van elk 256 bytes (paragraaf 1.2.) zullen we voor de 16 Kbyte blokken de Engelse uitdrukking page gebruiken.

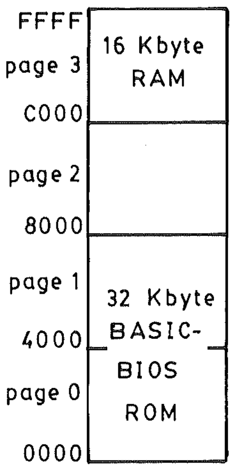


Fig. 63. Minimale indeling van de geheugenruimte.

De zogenaamde *"minimale configuratie"* zoals die voor de MSX- standaard is voorgeschreven, is weergegeven in figuur 63. page 0 is het onderste 16 Kbyte blok van de geheugenruimte. page 1, 2 en 3 komen in volgorde daar boven. Zoals in paragraaf 6.12 is vermeld zal de adresbus, direct na het inschakelen van de computer, het adres 0000 aanwijzen als eerste geheugenplaats waarvan de inhoud door de processor wordt gelezen. Dit is het startadres van het systeemprogramma. Het kan daarom niet anders of op deze plaats moet zich een ROM-geheugenplaats bevinden waarvan de inhoud bij het inschakelen aanwezig is. Het systeemprogramma vormt samen met de BASIC-interpreter bij de MXS-computer een uitgebreid programma dat een geheugenruimte van 32 Kbyte in beslag neemt, page 0 en page 1. Page 2 is bij de minimale uitvoering van de computer leeg. In page 3 vinden we een 16 Kbyte RAM-geheugenblok. In het RAM-geheugen worden de BASIC- programma's opgeslagen en eventueel ook de machinetaalprogramma's. Dat betekent nog niet dat page 3 geheel voor deze programma's beschikbaar is. Het systeemprogramma gebruikt *bij elke uitvoering* het bovenste gedeelte van page 3 voor het opslaan van diverse gegevens, de zogenaamde *"systeemwerkruimte"*. Dit gedeelte begint bij adres F381 zodat de bovenste geheugenplaats van het blok

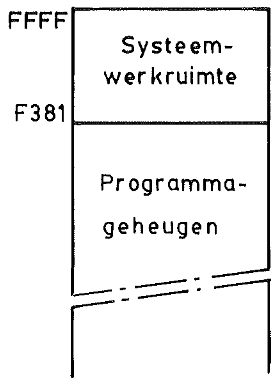


Fig. 64. De systeemwerkruimte.

dat vrij is voor programma's, zich op het adres F380 bevindt (figuur 64).
Overigens kunt u de bovenste grens van de geheugenruimte voor de programma's vastleggen met

CLEAR 200,adres.

Met het getal 200 hierin stelt u de maximum lengte van de stringruimte vast (in bytes) en met "adres" het hoogste adres dat voor de programma's beschikbaar is. De stringruimte neemt overigens ook een

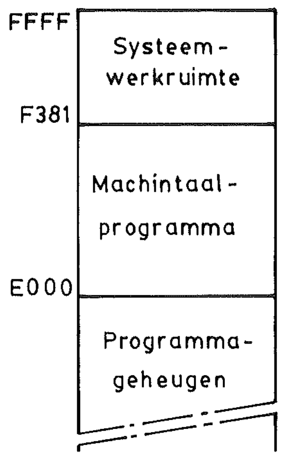


Fig. 65. Geheugenruimte voor een machinetaalprogramma.

gedeelte van het programmageheugen in beslag. Maakt u gebruik van een combinatie van een BASIC-programma en een machinetaalprogramma dan wordt het machinetaalprogramma in het algemeen in het bovenste gedeelte van het programmageheugen opgeslagen. Er moet dan ruimte voor dat programma worden gereserveerd. Wilt u bijvoorbeeld de ruimte van E000 tot en met F380 voor machinetaalprogramma's reserveren (figuur 65) gebruik dan

```
CLEAR 200,&HDFFF
```

Als u een grotere stringruimte nodig hebt dan moet u hiervoor uiteraard het juiste getal invoeren.
De bovenste grens van het programmageheugen is overigens vast gelegd in twee geheugenplaatsen van de systeemwerkruimte. Het zijn de geheugenplaatsen FC4A en FC4B. Probeer na het intoetsen van de bovenstaande regel maar eens:

```
A=PEEK(&HFC4A)+256*PEEK(&HFC4B)
PRINT HEX$(A)
```

Bij de grotere systemen zal ook page 2 gevuld zijn met RAM- geheugenelementen. Het laagste adres van het programmageheugen is dan 8000H.
Ook het aanvangsadres van een BASIC-programma is in de systeemwerkruimte vastgelegd. Hiervoor zijn de geheugenplaatsen F676 en F677.

```
A=PEEK(&HF676)+256*PEEK(&HF677)
PRINT HEX$(A)
```

Het resultaat zal steeds 1 groter zijn dan het laagste adres van het programmageheugen. Het werkelijke begin van een BASIC- programma is dan ook 1 geheugenplaats lager dan het resultaat. Een BASIC-programma heeft als eerste byte steeds een 0 en dat wordt door het startadres niet aangegeven.
Een adres, dat zoals hier in twee geheugenplaatsen is geschreven en dat het begin van een programma of een programmadeel aangeeft, wordt wel een *"vector"* genoemd. Elke vector is benoemd met een combinatie van maximaal zes karakters die, overeenkomstig een Mnemonic symbool, betrekking hebben op het doel van de vector. De vector die de top van de BASIC-geheugenruimte aangeeft is aangeduid met HIMEM (in de geheugenplaatsen FC4A en FC4B), de vector die het begin van een BASIC-programma markeert met TXTTAB (F676 en F677).
Het aanvangsadres van een BASIC-programma kan eventueel worden veranderd. Om alles goed te laten werken moet er echter iets meer geschieden dan alleen maar het veranderen van TXTTAB. Als voorbeeld verplaatsen we het begin van de geheugenruimte naar het adres D000H:

```
POKE &HF676,&H01:POKE &HF677,&HD0
POKE &HD000,0: NEW
```

Om te beginnen wordt TXTTAB veranderd in D001 (lage byte &H01, hoge byte &HD0). Dit is dus een plaats hoger dan het aanvangsadres van het geheugenblok. Daarna moet de eerste plaats van het geheugenblok 0 worden gemaakt (POKE &HD000,0). Als laatste is NEW nodig, ook als er zich geen programma in het geheugen bevindt. Boven een BASIC-programma in een geheugenruimte zijn een aantal blokken gereserveerd voor bijvoorbeeld de numerieke variabelen (VARTAB; F6C2 en F6C3), DIM array variabelen (ARYTAB; F6C4 en F6C5) enz. De vectoren die deze blokken aanwijzen worden wel *"pointers"* genoemd omdat ze geen constante inhoud hebben maar veranderen (in dit geval afhankelijk van de lengte van het BASIC-programma). Deze pointers krijgen de juiste waarde door NEW.

7.2. Slot-selectie.

Uit het bovenstaande moet u haast wel concluderen dat maximaal 32 Kbyte aan RAM-geheugenelementen in uw computer zijn aan te brengen en dat deze nog niet eens allemaal voor het programmeren ter beschikking staan. Het "28185 Bytes free" waarmee het beeldscherm u welkom heet bevestigt

dat vermoeden. En toch is het waar als een fabrikant zegt dat zijn computer is voorzien van 80 Kbyte RAM-geheugenelementen. In dat geval staat in het algemeen een blok van 16 Kbyte ter beschikking van de Vidio Display Processor (VDP). Deze verzorgt alles wat met het beeldscherm te maken heeft. Hij maakt een geschikt videosignaal, aangepast aan uw tv. Hij zorgt ervoor dat de juiste karakters op het scherm komen of de juiste grafische figuren, behandelt de werking van de sprites en dat allemaal in de door u gewenste kleuren. De VDP is een processor die geheugenelementen nodig heeft. Hiervoor dient de genoemde 16 Kbyte RAM die ook wel *"Video-RAM"* wordt genoemd. Deze is niet in de geheugenruimte van de Z80 opgenomen en wordt dan ook geadresseerd door de VDP. De VDP vormt in principe een computer in uw computer. De 16 Kbyte "VideoRAM" is bij elke MSX- computer aanwezig. Van de 80 Kbyte RAM-geheugenelementen blijft daardoor nog 64 Kbyte over om te worden gebruikt bij de Z80. Bij BASIC-programma's staan daarvan niet meer dan 32 Kbyte ter beschikking. Bij machinetaalprogrammeren kan eventueel echter de gehele 64 Kbyte RAM worden benut. Dan wordt de 32 Kbyte ROM voor de BASIC-interpreter en het systeemprogramma uitgeschakeld en een evengroot RAM-geheugenblok ingeschakeld.

Als u het Cartridge-slot van uw computer opent ziet u in de diepte de connector waarin u de contactstrip van de Cartridge kunt steken. Een Cartridge is in principe niets anders dan een "ingeblikte" printplaat waarop zich ROM-geheugenelementen kunnen bevinden met een programma, maar waarop ook in plaats van het ROM, RAM-geheugenelementen kunnen zijn aangebracht. Maximaal kunnen 64 Kbyte aan geheugenelementen in een Cartridge worden aangebracht. Afhankelijk van de uitvoering kunnen er ook meer van deze Cartridge-slots in uw computer zijn gemaakt. "Slot" is het Engelse woord voor "gleuf" en het wordt gebruikt voor de gleufvormige connector waarin de Cartridge kan worden gestoken. Als op de hoofdprint van uw computer geen geheugenelementen waren aangebracht dan zou een Cartridge met een 32 Kbyte BASIC ROM en nog 32 KByte RAM in een slot geplaatst kunnen worden. De computer kan dan van dit geheugenblok gebruikmaken. Op overeenkomstige wijze als de in- en uitgangspoorten moet de computer het slot activeren. De MSX-computer kan uit meerdere slots kiezen (indien aanwezig) en één van deze slots zal hij activeren. Het maximum is vier slots. Zou de computer vier slots bezitten dan zou er geen plaats meer zijn voor een 64 Kbyte geheugenblok op de hoofdprint. In werkelijkheid maakt het niet uit of een 64 Kbyte geheugenblok zich in een Cartridge bevindt die in een "slot" is gestoken. Er kan steeds maximaal uit vier *geheugenblokken* worden gekozen. Het selecteren van een 64 Kbyte geheugenblok wordt bij de MSX-computer *"slotselectie"* genoemd, ook als het geheugenblok zich niet in een slot bevindt maar op de hoofdprint. Figuur 66 geeft een voorstelling van de vier geheugenblokken die elk als "slot" zijn aangeduid en zijn genummerd van 0 tot en met 3.

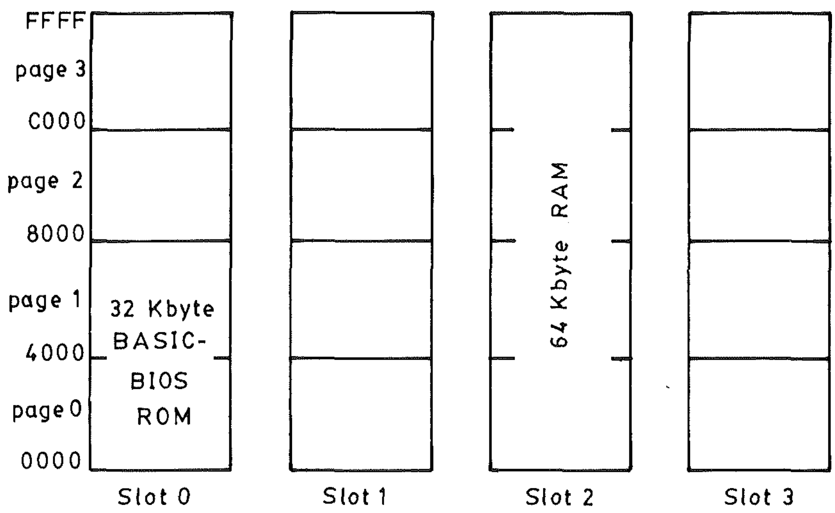


Fig. 66. Slotindeling "Goldstar FC-200".

Standaard neemt de BASIC ROM page 0 en page 1 van slot 0 in beslag. Bij een computer met een 16 Kbyte RAM (naast de vidioRAM) is page 3 van een van de vier slots bezet. Welk slot is afhankelijk van de fabrikant. Hier houdt dus de MSX-standaard op. Een computer met een 32 Kbyte RAM benut hiervoor page 2 en page 3 van een van de slots. Bij een computer met een 64 Kbyte geheugen is in het algemeen een van de slots 1, 2 of 3 volledig door het geheugen bezet. De pages 2 en 3 van slot 0 zijn dan niet van geheugenelementen voorzien. Hoe de slotindeling van uw computer is daar hoort de meegeleverde handleiding uitkomst over te geven. Fig. 66 toont de indeling bij de Goldstar FC-200.

Hoe de indeling van de slots ook is, de processor kan nooit meer dan 64 Kbyte geheugenruimte adresseren en het is daardoor ook niet mogelijk om te werken in dezelfde pages van twee verschillende slots. De computer kan de slots per page selecteren. Zo heeft de Goldstar FC-200 bij het inschakelen page 0 en page 1 van slot 0 geselecteerd en van het RAM-geheugenblok page 2 en page 3 van slot 2. Hiervan merkt u niets en het is alsof u werkt in een aaneengesloten geheugenblok van 64 Kbyte. Maakt u gebruik van BASIC-programma's (ook als u daarin een machinetaalprogramma hebt verwerkt) dan is dit de enige configuratie die mogelijk is, de situatie die optreedt als de desbetreffende computer wordt ingeschakeld. Omdat de BASIC ROM moet worden gebruikt zijn dan steeds page 0 en 1 in slot 0 geactiveerd. De RAM geheugenelementen die zich eventueel in page 0 en page 1 in een ander slot bevinden zijn dan onbereikbaar. Wordt echter uitsluitend met een machinetaalprogramma gewerkt, zodat de BASIC ROM niet wordt gebruikt, dan kan (als het programma meer geheugenruimte nodig heeft dan 32 Kbyte) door de juiste slotselectie van alle RAM- geheugenelementen worden gebruikgemaakt.

Het selecteren van de slots moet vanuit elk geheugenblok mogelijk zijn. Daarom is voor deze selectie gebruikgemaakt van een uitgangspoortregister, het zogenaamde *"slot select register"*. De in- en de uitgangspoorten bevinden zich niet in een slot maar vormen een zelfstandig geheugenblok dat door een IN of een OUT instructie wordt geactiveerd. Met het slotselectregister kan voor elke page worden gekozen welk slot moet worden geactiveerd, voor page 0 met de bits 0 en 1, voor page 1 met de bits 2 en 3, voor page 2 met de bits 4 en 5 en voor page 3 met de bits 6 en 7. Voor de Goldstar FC-200 is de aanvangssituatie van het slotselectregister aldus:

76	54	32	10	bitnummer.
10	10	00	00	slotselectie.
P3	P2	P1	P0	page-nummer.

Voor page 0 en page 1 is het slot 0 actief (de BASIC ROM is ingeschakeld) en voor page 2 en page 3 is slot 2 gekozen (voor de RAM-geheugenelementen die zich daarin bevinden). De twee bits voor elke page vormen het binaire getal van het slotnummer:

00 slotnummer 0
01 slotnummer 1
10 slotnummer 2
11 slotnummer 3

7.3. Het invoeren van machinetaalprogramma's.

De MSX-BASIC biedt u veel mogelijkheden bij het programmeren. Er is echter een belangrijk nadeel en dat is de traagheid. Elke regel van het machinetaalprogramma moet tijdens het doorlopen daarvan worden vertaald en dat kost tijd. Dit wordt pas goed duidelijk als er sprake moet zijn van bewegende schermbeelden. Het steeds opnieuw omzetten van de instructies door de interpreter neemt zoveel tijd in beslag dat daardoor de beelden slechts langzaam over het scherm kunnen bewegen. Daarvoor is een machinetaalprogramma dan ook veel meer geschikt. De snelheid daarvan is zo groot dat soms wel een vertraging moet worden ingelast. Door de vertragingsduur te regelen is ook de snelheid van de beelden op het scherm te regelen.

Het programmeren in machinetaal heeft echter ook zo zijn bezwaren. Voor een simpele werking van de computer zijn vaak een groot aantal machinetaalinstructies nodig zodat grote machinetaalroutines ontstaan.

Het programmeren in BASIC heeft zeker voordelen ten opzichte van het programmeren in machinetaal. Uiteindelijk is deze "hogere" programmeertaal nu juist daarvoor ontwikkeld. Andersom is het best mogelijk dat voor bepaalde werkingen alleen maar een machinetaalroutine kan worden gebruikt. Om de voordelen van beide methoden te kunnen benutten zijn er BASIC-instructies waarmee in een BASIC-programma een machinetaalprogramma kan worden ingelast. Ook is er een mogelijkheid om vanuit een BASIC-programma een machinetaalpro-

gramma op te bouwen. Dat houdt in dat een aantal geheugenplaatsen worden voorzien van codegetallen. Hiervoor dient de instructie

POKE AD,cg

Hierin is AD het adres van de geheugenplaats en cg het codegetal. AD en cg kunnen in elke talvorm worden ingevoerd, bijvoorbeeld:

POKE &HE012,&H5A
POKE &B1110000000010010,&B01011010
POKE 57362,90

Het hexadecimale talstelsel heeft bij het machinetaalprogrammeren de voorkeur.
De machinetaalprogramma's in dit boek zullen beperkt van lengte zijn. Ze zullen boven in de geheugenruimte gesitueerd zijn vanaf adres &HE000 tot adres &HF380. Deze ruimte kunt u reserveren met

CLEAR 200,&HDFFF

Binnen deze ruimte zullen steeds dezelfde startadressen worden gebruikt:

Programmastartadres: &HE000.
Variabelenstartadres: &HE800.
Top van de STACK: &HE7FF

Zoals bekend groeit de STACK naar beneden toe aan.
Als vanuit een BASIC-programma een machinetaalroutine moet worden aangeroepen dan moet eerst het aanvangsadres van deze routine worden gedefinieerd. Hiervoor is de instructie

DEF USR x = AD

Hierin is x een indexnummer (van 0 tot en met 9) en AD het startadres van het machinetaalprogramma. Er kunnen meerdere machinetaalroutines in een BASIC-programma worden opgenomen. Stel er zijn twee routines waarvan de eerste start op &HE000 en de tweede op &HE100, dan luiden de definities:

DEFUSR0 = &HE000
DEFUSR1 = &HE100

Op de juiste plaats in het BASIC-programma kan de machinetaalroutine worden aangeroepen met

D = USR x(P)

Hierin is D een variabelenaam waarvan het nut verderop in deze paragraaf zal worden verklaard, evenals P. Verder is x het indexnummer van de gewenste machinetaalroutine. Met

D = USR1(0)

wordt de machinetaalroutine met het startadres &HE100 aangeroepen. De machinetaalroutine moet als een subroutine zijn uitgevoerd, zodat na het doorlopen weer naar het BASIC-programma zal worden teruggesprongen. De machinetaalroutine zal daarvoor met een RET-instructie moeten worden afgesloten.
Als eerste voorbeeld van het opbouwen en het aanroepen van een machinetaalroutine vanuit een BASIC-programma volgt hier de aftrekking van twee zestienbits binaire getallen uit paragraaf 6.3.
De getallen zijn:

getal a: 01101010 00011101
getal b: 01001011 01101011

```
10 DEFUSR=&HE000
20 FOR AD=&HE000 TO &HE00E: READ A$
30 POKE AD,VAL("&H"+A$): NEXT
40 DATA 3E,1D:'     LD A,1D
50 DATA D6,6B:'     SUB 6B
60 DATA 32,00,E8:'  LD (E800),A
70 DATA 3E,6A:'     LD A,6A
80 DATA DE,4B:'     SBC A,4B
90 DATA 32,01,E8:'  LD (E801),A
100 DATA C9:'       RET
110 D=USR(0)
120 HB=PEEK(&HE801): LB=PEEK(&HE800)
130 PRINT HEX$(HB);HEX$(LB)
```

In regel 10 van het (BASIC-)programma wordt het startadres van de machinetaalroutine gedefinieerd. Omdat slechts één routine wordt aangeroepen kan het indexgetal worden weggelaten.
DEFUSR = &HE000 wordt door de computer gelezen als DEFUSR(0) = &HE000. De regels 20 en 30 bouwen het machinetaalprogramma op. Om de data niet steeds met het voorvoegsel "&H" te moeten intoetsen (de data wordt hexadecimaal ingevoerd) is de truc POKE AD,VAL("&H" + A$) toegepast. De regels 40 tot en met 100 geven de data die ingevoerd moet worden. Op elke regel staat een instructie van het machinetaalprogramma, afgesloten met de betekenis van de codegetallen in een REM-statement. Met behulp hiervan kunt u de codegetallen controleren met de instructieset. De instructievolgor-

de is geheel gelijk aan die welke de paragraaf 6.3. is gegeven. In regel 100 wordt de RET-instructie ingevoerd die de routine afsluit. Met D = USR(0) wordt de machinetaalroutine aangeroepen. Door de instructies op de regels 60 en 90 wordt het resultaat in de volgorde lage byte - hoge byte in het datageheugen opgeslagen. De PEEK-instructies in regel 120 lezen het resultaat terwijl dat in regel 130 op het scherm wordt getoond.

U kunt met de USR-functie elke gewenste variabele overbrengen van het BASIC-programma naar de desbetreffende machinetaalroutine. Probeer hiervoor eens het volgende programma:

```
10 POKE &HE000,&HC9:DEFUSR=&HE000
20 D=USR(1/7)
30 PRINT D
```

Het opgebouwde machinetaalprogramma is nu wel heel erg eenvoudig: niet meer dan RET (&HC9) op het adres &HE000. Door de functie D = USR(1/7) wordt eerst 1/7 uitgerekend en het resultaat hiervan wordt in een paar speciaal hiervoor gereserveerde geheugenplaatsen in de systeemwerkruimte (DAC, te beginnen op &HF7F6) opgeslagen. Daarna wordt de machinetaalroutine doorlopen. Dat betekent hier niet anders dan direct weer terug naar BASIC. Dan treedt opnieuw de functie D = USR(P) in werking. Nu wordt de variabele D gevuld met de waarde van de variabele die in DAC is opgeslagen. Regel 30 toont deze variabele. Nu heeft een programma als dit weinig meer nut dan het demonstreren van de werking van de USR(P)-functie. In werkelijkheid wordt de variabele in DAC eerst door de machinetaalroutine gebruikt en het eventuele resultaat weer terug in DAC geplaatst. Later ziet men door PRINT D het resultaat van de bewerking door de machinetaalroutine, op het scherm.
Het resultaat van de deling 1/7 is een gebroken getal. De computer geeft het antwoord met maar liefst 14 cijfers na de komma. De manier die de computer gebruikt om het getal in de geheugenplaatsen voor DAC op te slaan is die van de *"drijvende komma"* (floating point). Hierbij wordt het getal geschreven in de vorm van een mantissa en een exponent. Het getal is bij de "dubbele precisie" steeds veertien cijfers groot. Een getal als 1234.5678901234 wordt in de computer genoteerd als

$0.12345678901234 \times 10^4$

Hierin is 12345678901234 de mantissa en 4 de exponent. De computer slaat de mantissa op in de geheugenplaatsen F7F7 tot en met F7FD. Dit gebeurt in de BCD-code. Elke geheugenplaats kan twee cijfers bevatten en de zeven geheugenplaatsen zijn samen dus goed voor een mantissa van veertien cijfers. De exponent is in geheugenplaats &HF7F6 opgeslagen. Hiervan staan de zes rechtse bits ter beschikking (de bits 0 tot en met 5). Het grootste getal dat hierin kan worden opgeslagen is 3FH, 63 decimaal. Bit 6 van dit register is 1 en bit 7 wordt als tekenbit gebruikt. Bij een positief getal is bit 7 "0" en bij een negatief getal is bit 7 "1".

7	6	543210 bitnummer.
0	1	000000

Het resultaat van 1/7 moet dus in de registers van DAC als volgt zijn opgeslagen:

40 14285714285714

Het getal van de exponent in register F7F6 is nul. Omdat bit 6 "1" en bit 7 "0" is wordt de waarde van dit register 40H. De exponent wordt *niet* in BCD-code genoteerd maar gewoon binair. Het getal is daarmee te noteren als

$0.14285714285714 \times 10^0$

Na het uitvoeren van de USR(P)-functie wordt de inhoud van DAC gewijzigd, zodat later niet meer de inhoud van deze geheugenplaatsen is te controleren. Daarom gebruiken we het volgende programma:

```
10 DEFUSR=&HE000: A=1/7
20 FOR AD=&HE000 TO &HE00B: READ A$
30 POKE AD,VAL("&H"+A$): NEXT
40 DATA 21,F6,F7:' LD HL,F7F6
50 DATA 11,00,E8:' LD DE,E800
60 DATA 01,08,00:' LD BC,0008
70 DATA ED,B0:'    LDIR
80 DATA C9:'       RET
90 PRINT" "USR(A)
100 PRINT HEX$(PEEK(&HE800))" ";
110 FOR AD=&HE801 TO &HE807
120 PRINT HEX$(PEEK(AD));:NEXT
```

Met het machinetaalprogramma dat hierin wordt opgebouwd wordt een geheugenblokje van acht geheugenplaatsen vanaf F7F6 verplaatst naar het "datageheugen", te beginnen met E800. Dit gebeurt met de instructie LDIR op regel 70. Daarvoor moet HL

geladen worden met het adres van het brongeheugen (regel 40), moet DE geladen worden met het adres van het doelgeheugen (regel 50) en moet BC worden ingeschreven met 8 voor 8 geheugenplaatsen. Het aanroepen van de machinetaalroutine kan, zoals hier, ook gebeuren met

PRINT USR(A).

Ook nu wordt het resultaat afgedrukt van de bewerking door de machinetaalroutine. De regels 100 tot en met 120 laten nu het getal zien zoals dat in DAC was opgeslagen. Verander regel 10 van het programma nu eens in

```
10 DEFUSR=&HE000: A=-100/7
```

en RUN het opnieuw. U ziet dat nu de inhoud van register F7F6 gelijk is aan C2H:

7	6	543210 bitnummer.
1	1	000010 (C2H)

Dat het getal negatief is kunt u herkennen aan de waarde van bit 1. Een manier om de geheugenplaatsen van DAC te gebruiken demonstreert het volgende programma.

```
10 DEFUSR=&HE000
20 FOR AD=&HE000 TO &HE010: READ A$
30 POKE AD,VAL("&H"+A$): NEXT
40 DATA 3E,25:'      LD A,25
50 DATA D6,87:'      SUB 87
60 DATA 27:'         DAA
70 DATA 32,F8,F7:'   LD (F7F8),A
80 DATA 3E,71:'      LD A,71
90 DATA DE,36:'      SBC A,36
100 DATA 27:'        DAA
110 DATA 32,F9,F7:'  LD (F7F9),A
120 DATA 3E,02:'     LD A,02
130 DATA 32,63,F6:'  LD (F663),A
140 DATA C9:'        RET
150 D=USR(0)
160 PRINT HEX$(D)
```

Hiermee wordt de aftrekking uitgevoerd van de twee getallen van vier cijfers in de BCD-code, zoals beschreven in paragraaf 6.6. Waarschijnlijk is het gebruik van DAC in dit programma niet geheel zoals u zich dat had voorgesteld. Ten eerste wordt geheugenplaats F7F7, die de eerste twee cijfers van de mantissa moet bevatten, niet gebruikt. De lage byte van het resultaat wordt in F7F8 geladen (regel 70) en de hoge byte in F7F9 (regel 110). Verder wordt ook geheugenplaats F663 in het programma betrokken. De reden is dat nu een integer getal in DAC is geladen. Hiervoor worden alleen de geheugenplaatsen F7F8 en F7F9 gebruikt. Welk soort getal in DAC is geplaatst wordt aangegeven in geheugenplaats F663 (VALTYP). Voor een integervariabele moet hierin het getal 02 worden geplaatst (regels 120 en 130). De inhoud van VALTYP voor de diverse variabelen is:

Integer: 02H.
Single precision: 04H.
Double precision: 08H.
De geheugenplaatsen van DAC die worden gebruikt zijn:

Integer: F7F8 en F7F9.
Single precision: F7F6 t.e.m. F7F9.
Double precision: F7F6 t.e.m. F7FD.

Behalve numerieke variabelen kan ook een stringvariabele door middel van USR(P) naar een machinetaalprogramma worden overgebracht. In dit geval vinden we in VALTYP het getal 03. In de geheugenplaatsen F7F8 en F7F9 wordt in de volgorde hoge byte - lage byte een adres gegeven. Op dit adres begint een geheugenblokje met de gegevens van de string. Het eerste adres hiervan geeft de stringlengte. Het tweede en het derde adres geven in de volgorde lage byte - hoge byte de locatie van de string. Dat houdt in dat in de geheugenplaatsen F7F8 en F7F9 een adres wordt gegeven waar de gegevens van de string te vinden zijn. In het volgende programma wordt door USR(P) een string ingevoerd. Deze wordt door het machinetaalprogramma verplaatst waarna de string door het BASIC-programma weer zichtbaar wordt op het scherm.

```
10 DEF USR=&HE000
20 FOR AD=&HE000 TO &HE016: READ A$
30 POKE AD,VAL("&H"+A$): NEXT
40 DATA ED,5B,F8,F7:' LD DE,(F7F8)
50 DATA 1A:'          LD A,(DE)
60 DATA 4F:'          LD C,A
70 DATA 32,20,E8:'    LD (E820),A
80 DATA 13:'          INC DE
90 DATA 1A:'          LD A,(DE)
100 DATA 6F:'         LD L,A
110 DATA 13:'         INC DE
120 DATA 1A:'         LD A,(DE)
130 DATA 67:'         LD H,A
140 DATA 06,00:'      LD B,0
150 DATA 11,00,E8:'   LD DE,E800
160 DATA ED,B0:'      LDIR
170 DATA C9:'         RET
180 A$="Machinetaal op de MSX."
190 D$=USR(A$)
200 FOR X=0 TO PEEK(&HE20)+1
210 A=PEEK(&HE800+X)
220 PRINT CHR$(A);: NEXT
```

Voor het verplaatsen van de inhoud van het geheugenblok waarin de string is opgeslagen is het nodig dat het beginadres hiervan in HL en de lengte in BC is geladen. Om dat te bereiken gaan we wat manipuleren met de processorregisters. Eerst wordt door de data in regel 40 DE voorzien van het *adres* van de stringlengte. Dan wordt via de Accu (regel 50) de stringlengte zelf in register C geladen (regel 60). U ziet dat hierbij de indirecte adressering een grote rol speelt. Omdat de stringlengte ook later nog nodig is in het programma wordt deze tevens in geheugenplaats E820 opgeslagen (regel 70). Door in regel 80 DE met 1 te verhogen kunnen we in regel 90 de lage byte van het beginadres van de string in de Accu en later (regel 100) in L laden. Op dezelfde wijze wordt de hoge byte van het beginadres in H geladen. Om het registerpaar BC de juiste inhoud te geven moet B ''0'' gemaakt worden (regel 140). Als laatste moet DE nog voorzien worden van het adres van het doelregister (regel 150) waarna het verplaatsen kan aanvangen.
In regel 180 wordt de stringvariabele A$ van zijn inhoud voorzien. Dat kan natuurlijk ook op elke andere manier die met de MSX-BASIC mogelijk is. Voor het invoeren van deze string en het aanroepen van de machinetaalroutine moet nu

D$ = USR(A$)

worden toegepast. In regel 200 wordt de stringlengte gebruikt om in regel 220 de karakters weer op het scherm te laten printen.

Het invoeren van een machinetaalprogramma door middel van POKE-opdrachten in een BASIC-programma is nogal een tijdrovende geschiedenis. Eerst moet de instructievolgorde worden bepaald (dat moet natuurlijk altijd) en daarna moeten van alle instructies de operatiecoden worden samengesteld. Deze moeten dan ook nog als data worden ingetoetst. Als een programma wordt gestart dan zullen we ervaren dat bij een wat groter machinetaalprogramma de fouten niet zullen uitblijven. Een fout in een machinetaalprogramma betekent meestal dat de computer op ''tilt'' gaat zodat een reset of het uitschakelen van de voedingsspanning nog het enige is dat overblijft. Het verdient dan ook aanbeveling een ingevoerd programma eerst te SAVEN voordat het wordt gestart. Gaat u wat grotere machinetaalprogramma's gebruiken dan is het gewoon noodzakelijk om een *Assemblerprogramma* aan te schaffen. Met dit programma kunt u de instructies in *Assemblertaal* invoeren. Assemblertaal wil zeggen dat u voor de instructie de aanduidingen gebruikt zoals ze in de instructieset voorkomen. De assembler bepaalt dan de operatiecoden en berekent ook de adressen waarin deze coden moeten worden geladen. Het invoeren van het programma (het schrijven van de operatiecoden in de juiste geheugenplaatsen) voert de assembler ook zelf uit. Kies vooral voor een zogenaamde *''LABEL-assembler''*. Hiermee kunt u aan bepaalde instructieregels (op elke regel staat een volledige instructie) een naam (label) geven en bij sprongopdrachten verwijzen naar deze label. De assembler berekent dan zelf de spronggadressen. Een voorbeeld van een programma dat met een assembler is ingevoerd ziet u hieronder:

```
 1                        ;****************
 2                        ;* Het resetten *
 3                        ;*      van     *
 4                        ;*   registers  *
 5                        ;****************
 6
 7                                    ORG   0E000H
 8
 9 E000 2105E8            START:      LD    HL,0E805H
10 E003 3E00                          LD    A,0
11 E005 77                LOOP:       LD    (HL),A
12 E006 2D                            DEC   L
13 E007 20FC                          JR    NZ,LOOP
14 E009 C9                            RET
15                                    END
```

Het is de uitwerking van het programmavoorbeeld uit paragraaf 6.9. waarmee de geheugenplaatsen E801 tot en met E805 worden gereset. De listing van het programma is uit een aantal kolommen opgebouwd. De eerste kolom geeft de regelnummers. Deze zijn steeds met 1 oplopend. De volgende kolom geeft de adressen waarin de eerste byte van een instructie is geladen. In deze kolom hebben de getallen dus ook een oplopende waarde. Het verschil tussen de waarden is gelijk aan het aantal bytes dat de instructie lang is. In de derde kolom vindt u per instructie de codegetallen. De bytes zijn aaneengeschreven. Op elke regel staat een complete instructie. De gegevens in de derde kolom vormen de bytes die u bij toepassing van een BASIC-programma in de dataregels moet invoeren. De vierde kolom (vanaf regel 9) bevat de labels. In dit programma zijn dat START en LOOP. De vijfde kolom bevat de Mnemonicsymbolen terwijl de laatste kolom het doel- en het bronregister weergeeft.
Met behulp van een Assemblerprogramma is het mogelijk om machinetaalprogramma's van com-

mentaar te voorzien, overeenkomstig het REM-statement bij BASIC. Hiertoe dient het ";" teken aan het commentaar vooraf te gaan. In de regels 1 tot en met 5 is dit teken gebruikt om het programma van een titel te voorzien. Op regel 7 ziet u een zogenaamd *"pseudo operand"*. Het "ORG" op deze regel geeft aan op welk adres de eerste instructie moet beginnen. Het adres staat erachter vermeld. De vorm is 0E000H. In dit geval wordt het getal voorafgegaan door een 0. Dit is noodzakelijk bij alle *positieve* (hexadecimale) getallen waarvan het eerste cijfer acht of groter dan acht is. In dat geval is het hoogstwaardige bit van de binaire vorm van het getal "1" zodat de assembler het voor een negatief getal zou aanzien (een adres is altijd een positief getal!). Voor het invoeren van de regels 9 tot en met 15 hoeft u alleen maar de tekst in te voeren die in de kolommen 4, 5 en 6 staat vermeld, en wel in deze vorm. Een label (max. 6 karakters) moet steeds worden afgesloten met een ":" (START:). Het nieuwe regelnummer wordt steeds door de assembler op het scherm geprint nadat een regel met "RETURN" is afgesloten. Een sprongadres (regel 11) *moet* steeds van een label worden voorzien. Alle labels die in een programma gebruikt worden moeten verschillend zijn. Behalve letters mogen ook cijfers worden gebruikt (LOOP1, LOOP2 enz.). De spronggopdrachten in het programma moeten het doel van de sprong aangeven door de label te gebruiken (regel 13). Op de laatste regel *moet* de pseudo operand END komen. Zonder deze operand kan de assembler zijn werk niet doen. Wordt elke regel van het nodige commentaar voorzien dan komt het programma er als volgt uit te zien:

```
1                     ;****************
2                     ;* Het resetten *
3                     ;*     van      *
4                     ;*   registers  *
5                     ;****************
6
7                                ORG  0E000H
8                                LOAD 0E000H
9
10                    ;Hoogste adres in HL
11 E000 2105E8        START:     LD   HL,0E805H
12                    ;ACCU reseten
13 E003 3E00                     LD   A,0
14                    ;Het door HL gegeven
15                    ;adres resetten
16 E005 77            LOOP:      LD   (HL),A
17                    ;Voor volgend reg.
18 E006 2D                       DEC  L
19                    ;Terug voor volgend
20                    ;register
21 E007 20FC                     JR   NZ,LOOP
22                    ;Alles gereset
23 E009 C9                       RET
24                               END
```

Met behulp van het commentaar en de beschrijving in paragraaf 6.9. kunt u de werking ervan volgen. Op regel 8 ziet u nog een extra pseudo operand: LOAD. De bedoeling hiervan is dat de assembler niet alleen assembleert met als startadres E000 maar het ook in het geheugen schrijft, te beginnen met dit adres (in principe: beginnen op het adres dat na LOAD is gegeven).

Dit was een opsomming van een aantal punten die bij de diverse assemblerprogramma's in het algemeen gelijk zijn. Voor het verdere gebruik van de assembler moet ik u naar de gebruiksaanwijzing verwijzen.

De volgende machinetaalprogramma's in dit boek zullen steeds in deze vorm worden gegeven. Heeft u nog geen assemblerprogramma dan kunt u altijd voor het invoeren van de codegetallen een BASIC-programma gebruiken en de bytes uit kolom 3 als data invoeren op de manier die hiervoor is gedemonstreerd. U kunt de machinetaalroutine altijd vanuit een BASIC-programma aanroepen met de D=USR(P) functie.

7.4. Het Basic Input/Output System (BIOS).

Bij het beschrijven van de slotindeling is al gebleken dat de MSX-standaard niet elk onderdeel van de computer betreft. Desondanks moeten de diverse commando's (de BASIC-statements bijvoorbeeld) bij de diverse fabrikaten dezelfde werking hebben. Daarom zijn er in de BASIC-ROM geheugenruimte een aantal adressen gereserveerd waarop subroutines zijn aan te roepen. Deze brengen een voorgeschreven werking tot stand. Vooral bij handelingen die via de ingangs- of de uitgangspoorten verlopen is het belangrijk om van deze subroutines gebruik te maken. Als voorbeeld van een toepassing van de subroutines dient het volgende programma.

```
1                     ;****************
2                     ;* Lezen van het *
3                     ;*  slot select  *
4                     ;*    register   *
5                     ;****************
6                     ;
7                                ORG  0E000H
8                                LOAD 0E000H
9
10
11                    ;Aanroepen van RSLREG
12 E000 CD3801                   CALL 0138H
13                    ;Resultaat in DAC
14 E003 32F8F7                   LD   (0F7F8H),A
15                    ;Integer parameter
16 E006 3E02                     LD   A,2
17 E008 3263F6                   LD   (0F663H),A
18 E00B C9                       RET
19                               END
```

U kunt ermee vaststellen wat de waarde van het "slotselectreigster" is (paragraaf 7.2.) bij BASIC-programmeren. Dat geeft u inzicht in de gebruikte slots.

Met de volgende BASIC-regel kunt u de waarde van het register op het scherm laten printen:

DEFUSR = &HE000:D = USR(0):PRINT BIN$(D)

Het volgende programma heeft dezelfde werking. Het demonstreert nog een gebruiksmogelijkheid van de assembler. In de regels 10 en 11 wordt met de pseudo operand EQU een variabelenaam toegewezen aan een geheugenadres. Later wordt in het programma in plaats van het adres, de variabelenaam gebruikt (regels 17 en 20).

```
1                     ;*****************
2                     ;* Lezen van het *
3                     ;*  slot select  *
4                     ;*    register   *
5                     ;*****************
6                     ;
7                             ORG  0E000H
8                             LOAD 0E000H
9
10                    DAC2:   EQU  0F7F8H
11                    VALTYP: EQU  0F663H
12
13
14                    ;Aanroepen van RSLREG
15 E000 CD3801                CALL 0138H
16                    ;Resultaat in DAC
17 E003 32F8F7                LD   (DAC2),A
18                    ;Integer parameter
19 E006 3E02                  LD   A,2
20 E008 3263F6                LD   (VALTYP),A
21 E00B C9                    RET
22                            END
```

7.5. Het gebruik van de "HOOK".

In paragraaf 6.10. is de instructie LDIR beschreven. Hierbij is gesteld dat bij overlappende blokken het doelregister een lager adres moet hebben dan het bronregister omdat anders data overschreven wordt die nog had moeten worden uitgelezen. Een uitzondering is als een heel blok met dezelfde data moet worden gevuld. Stel dat het doelregister (aangegeven door DE) precies een plaats hoger is dan het bronregister (aangegeven door HL) dan wordt door LDIR het gehele blok met de data van het eerste register gevuld (figuur 67). Na elke keer dat data is overgebracht worden HL en DE beide met 1 verhoogd zodat het zojuist ingeschreven register het bronregister wordt en dezelfde data weer een stapje hoger wordt geplaatst.

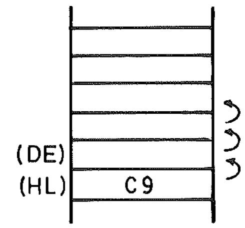


Fig. 67. Het vullen van een geheugenblok met C9.

Deze methode is toegepast om direct na het inschakelen van de computer een blok in het bovenste gedeelte van de systeemwerkruimte (van FD9A tot en met FFC9) te vullen met C9, de instructiecode voor RET. Een aantal routines uit de BASIC-ROM maken gebruik van deze geheugenplaatsen. Vaak in het begin -soms als eerste instructie- van de desbetreffende routine vinden we een CAL-instructie die een adres in het genoemde blok aanroept. Hier staat dan de instructie RET die de processor onmiddelijk weer terug laat gaan naar de ROM-routine. Het aangeroepen adres wordt een *"HOOK"* genoemd. Omdat deze zich in een RAM-geheugenblok bevindt is het mogelijk om de inhoud van dat adres te wijzigen en in te grijpen in de desbetreffende subroutine. Voor dat ingrijpen staan vijf geheugenplaatsen ter beschikking. Steeds vijf geheugenplaatsen verder vinden we namelijk de HOOK van een andere ROM-routine. Het is daarom steeds mogelijk de instructie C9 te vervangen door een CAL- of een JP-instructie naar een subroutine die ons in staat stelt de routine een extra functie te geven.
In het volgende programma worden drie HOOKS gebruikt. De eerste is die van de CHGET subroutine. Deze start op het adres 009F en wordt steeds aangeroepen als er een toets is ingedrukt. De HOOK bevindt zich op het adres FDC2. Door hier de instructie

JP 00C0

in te laden wordt steeds als er een toets is ingedrukt naar de routine op het adres 00C0 gesprongen. Dit is de routine voor BEEP zodat bij elke toetsaanslag het BEEP-toontje wordt gehoord.
Door het programma wordt in de regels 11 en 12 het adres 00C0 op zijn plaats gebracht (in de geheugenplaatsen FDC3 en FDC4). De HOOK zelf is nog niet van C3 voorzien, dat gebeurt later. De tweede HOOK is die van de routine voor het commando SET. Er zijn een aantal routines die reageren op commando's die bestemd zijn voor randapparaten.

Zijn deze niet aangesloten dan heeft het intoetsen van een dergelijk commando een foutmelding tot gevolg. De desbetreffende routine met de bijbehorende HOOK wordt echter wel aangeroepen. De HOOK voor de routine voor SET bevindt zich op het adres FDF4. In de geheugenplaatsen FDF5 en FDF6 wordt eerst door de regels 14 en 15 het adres E01C geladen, een adres van het programma zelf dus. Ook de HOOK op FE0D wordt van een adres (E020) voorzien (regels 17 en 18). Deze HOOK wordt aangeroepen door de routine voor CMD. Uiteindelijk worden de HOOKS FDF4 en FE0D van de instructie C3 voorzien. De HOOK op FDC2 heeft echter nog steeds de oorspronkelijke waarde. Als we met

DEFUSR = &HE000: D = USR(0)

het machinetaalprogramma in werking hebben gesteld dan hebben we er twee nieuwe commando's bij gekregen: SET en CMD. Toetsen we SET in dan heeft dat tot gevolg dat via de HOOK ons programma op regel 26 wordt aangeroepen. In deze regel en in de regels 27 en 30 wordt nu in de HOOK op FDC2 de instructie C3 geladen. Het gevolg is dat elke keer als een toets wordt aangeslagen het BEEP- toontje te horen is. Toetsen we daarna het commando CMD in dan wordt in de regels 29 en 30 C9 in de HOOK geladen en is de rust weergekeerd.

```
 1                  ;************
 2                  ;* Set BEEP *
 3                  ;*   toon   *
 4                  ;************
 5
 6                                ORG  OE000H
 7                                LOAD OE000H
 8
 9
10                  ;Adres van BEEP rout. in H-CHGE
11 E000 21C000 START:             LD   HL,00C0H
12 E003 22C3FD                    LD   (OFDC3H),HL
13                  ;Adres van SETS in H-SETS
14 E006 211CE0                    LD   HL,OE01CH
15 E009 22F5FD                    LD   (OFDF5H),HL
16                  ;Adres van CMD in H-CMD
17 E00C 2120E0                    LD   HL,OE020H
18 E00F 220EFE                    LD   (OFE0EH),HL
19                  ;JP-instr. in H-SETS en H-CMD
20 E012 3EC3                      LD   A,OC3H
21 E014 32F4FD                    LD   (OFDF4H),A
22 E017 320DFE                    LD   (OFE0DH),A
23                  ;Einde hoofdprogramma.
24 E01A 1809                      JR   EIND
25                  ;Bij SET: JP in H-CHGE
26 E01C 3EC3   SETS:              LD   A,OC3H
27 E01E 1802                      JR   STORE
28                  ;Bij CMD: RET in H-CHGE
29 E020 3EC9   CMD:               LD   A,OC9H
30 E022 32C2FD STORE:             LD   (OFDC2H),A
31 E025 C9     EIND:              RET
32                                END
```

In lijst 20 is een selectie van een aantal HOOKS gegeven, met bij elke hoek het adres van de ROM-routine die de HOOK aanroept.

Lijst 20. Een aantal HOOKS.

Naam	Adres.	CALL adr.	Omschrijving.
H-ATTR	FE1C	7C43	Attribute rout.
H-CHGE	FDC2	10CE	Character get
H-CHPU	FDA4	08C0	Character put
H-CMD	FE0D	7C34	Command CMD
H-COPY	FE08	7C2F	COPY files
H-CVD	FE49	7C70	Convert dbl
H-CVI	FE3F	7C66	Convert int.
H-CVS	FE44	7C6B	Convert sgn.
H-DSKF	FE12	7C39	Disk free
H-DSKI	FE17	7C3E	Disk input
H-DSKO	FDEF	7C16	Disk output
H-DSPC	FDA9	09E6	Display cursor
H-DSPF	FDB3	0B2B	Dspl. funct.key
H-ERAC	FDAE	0A33	Erase cursor
H-ERAF	FDB8	0B15	Erase funct.key
H-FIEL	FE2B	7C52	FIELD
H-FORM	FFAC	148E	FORMAT
H-INIP	FDC7	071E	Initialise patt
H-INLI	FDE5	23D5	Line input
H-IPL	FE03	7C2A	Init.progr.load
H-KEYC	FDCC	1025	Key coder
H-KEYI	FD9A	0C4A	Interrupt handler
H-KILL	FDFE	7C25	KILL file
H-KYEA	FDD1	0F10	Key easy
H-LPTO	FFB6	085D	Line printer out
H-LPTS	FFBB	0884	Line print.status
H-LSET	FE21	7C48	Left set
H-MKD$	FE3A	7C61	Make double
H-MKI$	FE30	7C57	Make int.
H-MKS$	FE35	7C5C	Make single
H-NAME	FDF9	7C20	Rename
H-NMI	FDD6	1398	NMI (interrupt)
H-ONGO	FDEA	7810	On device gotó
H-OUTD	FEE4	1B46	OUT DO
H-PHYD	FFA7	148A	Phys. disc i/o
H-PINL	FDDB	23BF	Program line
H-PLAY	FFC5	73E5	PLAY
H-QINL	FDE0	23CC	Question mark and input line
H-RSET	FE26	7C4D	Right set
H-SCRE	FFC0	79CC	SCREEN
H-SETS	FDF4	7C1B	SET attributes
H-TIMI	FD9F	0C53	Interrupt handler
H-TOTE	FDBD	0842	Force screen to text mode
H-WIDT	FF84	51CC	WIDTH

7.6. Het SAVEN en LOADEN van machinetaalroutines.

Het SAVEN van een machinetaalroutine houdt in principe niets anders in dan dat de inhoud van een geheugenblok naar het externe geheugen (cassette of disk) wordt geschreven. Hiervoor kan het commando BSAVE worden gebruikt. Aan dit commando moet nog worden toegevoegd het beginadres van het programma, het eindadres van het programma en eventueel het startadres. Het beginadres is het adres

van de eerste geheugenplaats. Het eindadres is het eerste adres direct boven het geheugenblok. Wordt het adres van de laatste geheugenplaats als eindadres ingevoerd dan wordt dit net niet weggeschreven. Bij sommige programma's is het eerste adres niet het startadres. Het startadres bevindt zich dan wat verderop in het programma. Bij het laden van een machinetaalprogramma vanuit een extern geheugen kan een commando worden gegeven dat het direct wordt gestart. In dat geval is het nodig bij het wegschrijven naar de disk of tape het startadres te vermelden. Wordt het startadres weggelaten dan wordt het beginadres als het startadres beschouwd. Het in de vorige paragraaf gegeven programma zou met het volgende commando naar de cassette kunnen worden geschreven:

BSAVE"CASS:STREEP",
&HE000,&HE026,&HE000

De aanduiding van het randapparaat "CAS:" kan worden weggelaten als u niets anders dan de cassetterecorder als externe geheugen hebt aangesloten. Het invoeren van een naam (maximaal zes karakters) is noodzakelijk: BSAVE"STBEEP". Het beginadres is &HE000 en het eindadres &HE026. Het startadres &HE000 als laatste ingevoerde gegeven zou hier weggelaten kunnen worden, omdat dit gelijk is aan het beginadres. Later kan het programma weer in de computer worden gelezen met

BLOAD"CASS:STBEEP"

Als u BLOAD"CASS:" invoert, zonder een programmanaam dus, zal het eerstvolgende blok dat met BSAVE naar de band is geschreven, in de computer worden geladen. Heeft u alleen de cassetterecorder op de computer aangesloten dan werkt BLOAD"STBEEP" ook. Indien u de gegevens van de band in een ander gedeelte van het geheugen wilt laden dan waaruit het afkomstig is dan dient u het nieuwe beginadres in te voeren. Bijvoorbeeld:

BLOAD"STBEEP",&HD000

Programma's waarin sprongen voorkomen in de absolute extended addressing mode (JP) moeten steeds in hetzelfde geheugenblok worden geladen omdat anders de sprongadressen niet meer kloppen. Dat is ook het geval met routines die door een ander programmadeel worden aangeroepen (zoals in dit voorbeeld door een HOOK: SET EN CMD). Programma's die geheel zelfstandig zijn en waarbinnen slechts relatieve sprongadressen voorkomen (JR) kunnen in een ander geheugenblok worden geplaatst dan waar ze oorspronkelijk vandaan kwamen. In plaats van het nieuwe adres in te voeren kan het ook zo:

BLOAD"STBEEP",-&H1000

Eerder is al opgemerkt dat bij het commando voor het laden van een machinetaalprogramma uit een extern geheugen een toevoeging kan worden gegeven waarmee het programma direct na het laden wordt gestart. Het aanroepen vanuit BASIC met D=USR(0) is dan niet meer nodig. Voor deze toevoeging wordt de letter R gebruikt:

BLOAD"STBEEP",R

Het programma start op het adres dat bij BSAVE is gedefinieerd. Verder zijn de commando's BSAVE en BLOAD, in de vorm zoals ze hier zijn beschreven, volledig toepasbaar in een BASIC-programma.

8. Bijzonderheden van BASIC

8.1. BASIC-regels in het geheugen.

Soms worden veranderingen aangebracht in een BASIC-programma zoals zich dat in het programmageheugen bevindt. Dat dat alleen maar met een machinetaalprogramma kan plaatsvinden zal duidelijk zijn. Een voorbeeld hiervan zal in de volgende paragraaf worden gegeven. Om veranderingen te kunnen aanbrengen zullen we toch eerst moeten weten op welke manier de BASIC-regels in het geheugen zijn opgeslagen.

De eerste programmaregel vangt aan op het adres dat door TXTTAB wordt aangegeven (bij 64 Kbyte computers normaal op 8001H). De regels worden gescheiden door een geheugenplaats die met 00H is gevuld. Hierdoor zijn de diverse regels van elkaar te onderscheiden. De eerste twee geheugenplaatsen van *elke* regel bevatten een adres. Dit adres geeft in hexadecimale vorm (zoals gebruikelijk bij adressen) de geheugenplaats aan waar de *eerstvolgende* BASIC-regel aanvangt. In de twee volgende geheugenplaatsen vinden we het regelnummer. Hiervoor zijn twee geheugenplaatsen nodig omdat het grootste regelnummer 65529 kan zijn. Ook het regelnummer is hexadecimaal in deze geheugenplaatsen opgeslagen. Dat betekent dat regelnummers kleiner dan 256 slechts één geheugenplaats nodig hebben (alleen de lage byte) zodat de tweede geheugenplaats nul is. Omdat we weten dat de desbetreffende geheugenplaats bij een regelnummer hoort zal deze nul niet als de afscheiding tussen twee regels worden herkend. Direct na het regelnummer volgt het eerste BASIC- statement. Het statement is steeds in de vorm van een codegetal in het geheugen opgeslagen. Dit codegetal wordt een *''TOKEN''* genoemd. Omdat dat een zeer specifieke aanduiding is, is het beter hier niet de vertaling (teken) voor te gebruiken. Lijst 21 geeft voor alle MSX-BASIC statements de tokens. De codegetallen in deze lijst die kleiner zijn dan 31H worden eerst vermeerderd met 80H en voorafgegaan door FFH in de BASIC-geheugenruimte geplaatst. Zo heeft het statement ''ABS'' 06H als token. In de BASIC-regel in het geheugen wordt echter hiervoor FFH 86H genoteerd. Het token voor ''ELSE'' is A1H. In de BASIC-regel wordt dit (als enig token) voorafgegaan door 3AH, het token voor '':''.

Ook andere tekens, zoals bewerkingstekens, die niet in een string zijn geplaatst en die bijvoorbeeld betrekking hebben op bewerkingen met variabelen, worden met een token genoteerd (lijst 22). De codegetallen in deze lijst worden onveranderd in de BASIC-regel geplaatst, met uitzondering van het token voor ''''. Dit wordt gebruikt om een REM-statement aan te geven en in de BASIC-regel genoteerd als 3AH 8FH E6H.

Alle verdere tekens die in een BASIC-regel kunnen voorkomen, zoals de namen van variabelen en karakters in een string, worden door middel van het bijbehorende ASCII-codegetal in het geheugen geplaatst. Het resultaat van het volgende programma laat u de codegetallen van de eerste regel van dit programma zien:

```
10 PRINT"1,A"
20 AD=PEEK(&HF676)+256*PEEK(&HF677)
30 FOR X=0 TO 15
40 PRINT HEX$(PEEK(AD+X))" ";
50 NEXT
```

Behalve het resultaat van de eerste regel (PRINT''1,A'') ziet u ook de volgende regel op uw scherm:

```
C 80 A 0 91 22 31 2C 41 22 0 27 80 14 0 41
```

Deze regel is tot stand gekomen door de programmaregels 20 tot en met 50. In regel 20 wordt het startadres van het BASIC-programma bepaald (TXTTAB = &HF676). Daarna worden de eerste 16 codetekens op het scherm geprint. Omdat hier een 64 Kbyte machine is gebruikt is het eerste adres 8001. Volgens de eerste twee afgedrukte bytes is het adres van de tweede regel: 800C (0C is weergegeven als C). Op dit adres is het getal 27 geladen. Het is het adres direct volgend op de scheidingsnul. De volgende twee bytes geven het regelnummer: $000A_{hex} = 10_{dec}$. Nu komt het token voor PRINT; 91H, het token voor ''; 22 en de ASCII coden voor 1,A. Als laatste wordt de regel afgesloten met het token voor ''; 22. De volgende byte is 00H, het scheidingsteken waarna de volgende regel begint.

Commando	TOKEN	Commando	TOKEN	Commando	TOKEN	Commando	TOKEN
AUTO	A9	EXP	0B	END	81	MKI$	2E
ABS	06	FOR	82	EOF	2B	MKS$	2F
AND	F6	FIELD	B1	EQV	F9	MOD	FB
ASC	15	FILES	B7	ERASE	A5	MOTOR	CE
ATN	0E	FIX	21	ERL	E1	NAME	D3
ATTR$	E9	FN	DE	ERR	E2	NEW	94
BASE	C9	FPOS	27	ERROR	A6	NEXT	83
BEEP	C0	FRE	0F	NOT	E0	SGN	04
BIN$	1D	GET	B2	OCT&	1A	SIN	09
BSAVE	D0	GOSUB	8D	OFF	EB	SOUND	C4
CDBL	20	GOTO	89	ON	95	SPACE$	19
CHR$	16	HEX$	1B	OPEN	B0	SPC(	DF
CINT	1E	IF	8B	OR	F7	SPRITE	C7
CIRCLE	BC	IMP	FA	OUT	9C	SQR	07
CLEAR	92	INKEY$	EC	PAD	25	STEP	DC
CLODE	9B	INP	10	PAINT	BF	STICK	22
CLOSE	B4	INPUT	85	PDL	24	STOP	90
CLS	9F	INSTR	E5	PEEK	17	STR$	13
CMD	D7	INT	05	PLAY	C1	STRIG	23
COLOR	BD	IPL	D5	POKE	98	STRING$	E3
CONT	99	KILL	D4	POINT	ED	SWAP	A4
COPY	D6	KEY	CC	POS	11	TAB(	BD
COS	0C	LEFT$	01	PRESET	C3	TAN	0D
CSAVE	9A	LEN	12	PRINT	91	THEN	DA
CSNG	1F	LET	88	PSET	C2	TIME	CB
CSRLIN	E8	LFILES	BB	PUT	B3	TO	D9
CVD	2A	LINE	AF	READ	87	TRON	A2
CVI	28	LIST	93	REM	8F	USING	E4
DATA	84	LLIST	9E	RENUM	AA	USR	DD
DEF	97	LOAD	B5	RESTORE	8C	VAL	14
DEFDBL	AE	LOC	2C	RESUME	A7	VARPTR	E7
DEFINT	AC	LOCATE	D8	RETURN	8E	VDP	C8
DEFSGN	AD	LOF	2D	RIGHT$	02	VPEEK	18
DEFSTR	AB	LOG	0A	RND	08	VPOKE	C6
DELETE	A8	LPOS	1C	RSET	B9	WAIT	96
DIM	86	LPRINT	9D	SAVE	BA	WIDTH	A0
DRAW	BE	LSET	B8	SCREEN	C5	XOR	F8
DSKF	97	MAX	CD	SET	D2		
DSKI$	EA	MERGE	B6				
DSKO$	D1	MID$	03				
ELSE	A1	MKD$	30				

Lijst 21. TOKENS voor de BASIC-statements.

Teken	TOKEN
+	F1
-	F2
*	F3
/	F4
^	F5
<	F0
>	EE
=	EF
\	FC
'	E6
:	3A
;	3B
,	2C
!	21
#	23
$	24
%	25
"	22

Lijst 22. TOKENS voor programmatekens.

8.2. Bewerking van een BASIC-programma.

Een voorbeeld van het bewerken van een BASIC-programma geeft de volgende routine. Het is een wat langer programma dat in gedeelten zal worden besproken. De losse delen kunt u als een geheel in de computer laden. Als u van band of schijf een BASIC-programma in de computer laadt dan komen daarin vaak ook regels in voor met een REM-statement. De bedoeling daarvan zal u bekend zijn. Ze moeten het u mogelijk maken om het doel van de statements en het verloop van de programma's te kunnen volgen. Een nadeel van de REM-statements is echter dat ze geheugenruimte in beslag nemen en vaak vertragend werken op het verloop van het programma. Het volgende programma maakt het mogelijk om deze REM statements uit het BASIC-programma te verwijderen. Eerst wordt dit machinetaalprogramma in de computer geladen. Opdat u het ook daadwerkelijk zou kunnen gaan gebruiken is het zo hoog mogelijk in de geheugenruimte gesitueerd. U kunt het afschermen van de BASIC-programma's met

CLEAR 200,&HF2FF

Als het machinetaalprogramma op zijn plaats is kan het BASIC- programma worden geladen. Voordat dit wordt gestart moet de machinetaalroutine worden aangeroepen, eventueel met

DEFUSR=&HF300: D=USR(0)

Hiervoor kan echter ook een zelfgemaakt commando dienen. Nu is het niet zo moeilijk om de REM-statements in een BASIC-programma op te zoeken en te verwijderen, maar de regels van het programma moeten een gesloten geheel zijn in de geheugenruimte. De gaten die zouden ontstaan door het verwijderen van de REM-statements moeten worden voorkomen door het verplaatsen van de regels die hierna komen. Daarom wordt het registerpaar DE gebruikt als pointer. Deze geeft steeds het begin aan waar de volgende regel in het gecorrigeerde programma moet beginnen. De regels 19 tot en met 21 van het eerste deel van het programma dienen voor het initiëren:

```
1                     ;*****************
2                     ;* REM-destroyer *
3                     ;*****************
4
5
6                     TXTTAB:    EQU  0F676H
7                     NRGNRL:    EQU  0F601H
8                     NRGNRH:    EQU  0F602H
9                     SAVADL:    EQU  0F603H
10                    SAVADH:    EQU  0F604H
11
12
13                               ORG  0F300H
14                               LOAD 0F300H
15
16
17                    ;Initieren
18                    ;Bepaal begin BASIC-progr.
19 F300 2A76F6        INIT:      LD   HL,(TXTTAB)
20 F303 2201F6                   LD   (NRGNRL),HL
21 F306 ED5B01F6                 LD   DE,(NRGNRL)
22
```

Eerst wordt het begin van het BASIC-programma bepaald en in HL geladen. De geheugenplaatsen NRGNRL en NRGNRH zullen steeds voorzien zijn van het adres van de eerstvolgende BASIC regel die nog moet worden behandeld en dat is op dit moment nog de eerste regel. Verder moet ook DE van dit adres worden voorzien. Er is ook nog een stel geheugenplaatsen nodig voor het tijdelijk bewaren van een adres (SAVADL en SAVADH). Voor NRGNR en SAVAD zijn geheugenplaatsen gekozen in de systeemwerkruimte (regels 6 tot en met 10). Als *niet* vanuit BASIC wordt gewerkt maar een geheel zelfstandig machinetaalprogramma wordt gebruikt dan dient in het initieerdeel ook de plaats van de STACK te worden aangewezen. Hiervoor moet de Stackpointer met dit adres worden geladen. In dit geval laten we de STACK op de plaats die ook bij het BASIC- programma wordt gebruikt.

Eerst moet het programma de lengte van de BASIC regel bepalen die nog moet worden bewerkt. Dit moet bij elke volgende regel gebeuren en daarom wordt steeds naar de programmaregel 24 teruggesprongen. Hierin moet eerst HL van het adres van deze (nieuwe) regel worden voorzien. Dit adres moet voorlopig worden bewaard (regel 25). Ook de inhoud van DE moet worden bewaard. De eerste twee bytes van deze BASIC regel bevatten het adres van de volgende BASIC regel. Dit adres wordt opgeslagen in NRGNR (regels 28 tot en met 32). Dan weten we wat de plaats is van de volgende BASIC-regel als we aan het eind van het programma weer teruggaan naar regel 24.

Omdat HL het adres bevat van het begin van de regel en DE het adres bevat van het begin van de volgende regel kunnen we de regellengte berekenen door de twee waarden van elkaar af te trekken. DE bevat de grootste waarde en omdat slechts de in-

houd van DE van die van HL kan worden afgetrokken moet eerst de verwisseling EX DE,HL plaatsvinden. De aftrekking vindt plaats met SBC, zodat eerst het Carrybit nul moet worden (regel 36). Uiteindelijk wordt de regellengte vanuit HL overgebracht in BC. Bij het verlaten van dit programmadeel bevindt zich het adres waar de volgende BASIC-regel moet komen in DE, is het adres van van de te behandelen BASIC regel in HL en is de lengte van deze regel in BC geladen.

```
23                    ;Bepaal lengte BASIC-regel
24 F30A 2A01F6        BRGLL:      LD   HL,(NRGNRL)
25 F30D E5                        PUSH HL
26 F30E D5                        PUSH DE
27                    ;Adres nieuwe regel in DE
28 F30F 5E                        LD   E,(HL)
29 F310 23                        INC  HL
30 F311 56                        LD   D,(HL)
31                    ;Bewaar adres nieuwe regel
32 F312 ED5301F6                  LD   (NRGNRL),DE
33                    ;Bereken regellengte
34 F316 13                        INC  DE
35 F317 EB                        EX   DE,HL
36 F318 A7                        AND  A
37 F319 ED52                      SBC  HL,DE
38 F31B 2B                        DEC  HL
39                    ;Regellengte in BC
40 F31C 4D                        LD   C,L
41 F31D 44                        LD   B,H
42                    ;Beginadres (doel) in DE
43 F31E D1                        POP  DE
44                    ;Beginadres (bron) in HL
45 F31F E1                        POP  HL
46
```

Het volgende programmadeel is voor het verplaatsen van de BASIC regel. Het bronadres bevindt zich in HL, het doeladres in DE en de regellengte in BC. Het doeladres en de regellengte mogen daarbij niet verloren gaan en worden daarom in de STACK opgeborgen. Het LDIR op regel 53 heeft het verplaatsen tot gevolg. U zult misschien zeggen dat het verplaatsen helemaal niet nodig is zolang er nog geen REM-statement is verwijderd. In dat geval is de inhoud van HL gelijk aan die van DE en houdt het verplaatsen in dat de bron en het doel samenvallen. In werkelijkheid wordt er dan niet verplaatst. Aan het eind van dit programmadeel bevat DE het eindadres + 1 van de verplaatste regel (dat is het beginadres van de volgende regel), BC de regellengte en HL het beginadres van de verplaatste regel. Dit is met PUSH DE (regel 49) in de STACK geplaatst en in regel 57 met POP HL weer opgehaald en in HL geladen. Dit adres wordt aan het begin van het volgende programmadeel in de STACK bewaard.

```
47                    ;Verplaats BASIC-regel
48                    ;Beginadres doel bewaren
49 F320 D5            VRPL:       PUSH DE
50                    ;Regellengte bewaren
51 F321 C5                        PUSH BC
52                    ;Verplaats de regel
53 F322 EDB0                      LDIR
54                    ;regellengte in BC
55 F324 C1                        POP  BC
56                    ;Beginadres in HL
57 F325 E1                        POP  HL
58
59                    ;Zoek REM-statement
60                    ;Bewaar beginadres
61 F326 E5            ZKREM:      PUSH HL
62                    ;HL en teller op statement
63 F327 3E04                      LD   A,04H
64 F329 23            LOOP:       INC  HL
65 F32A 0B                        DEC  BC
66 F32B 3D                        DEC  A
67 F32C 20FB                      JR   NZ,LOOP
68                    ;Token voor REM
69 F32E 3E8F          ZKREM0:     LD   A,08FH
70                    ;Token voor REM in (HL)?
71 F330 BE                        CP   (HL)
72 F331 2808                      JR   Z,ZKREM1
73                    ;Geen REM, zoek ":"
74                    ;Token voor ":"
75 F333 3E3A                      LD   A,3AH
76 F335 EDB1                      CPIR
77                    ;Einde regel, regel afsl.
78 F337 200A                      JR   NZ,ZKREM3
79                    ;":" gevonden
80 F339 28F3                      JR   Z,ZKREM0
81                    ;Token voor apostrof
82 F33B 3EE6          ZKREM1:     LD   A,0E6H
83 F33D 23                        INC  HL
84 F33E BE                        CP   (HL)
85                    ;Geen apostrof, geen DEC
86 F33F 2001                      JR   NZ,ZKREM2
87                    ;Sluit regel af met "0"
88 F341 2B                        DEC  HL
89 F342 2B            ZKREM2:     DEC  HL
90                    ;Maak ACCU 0
91 F343 AF            ZKREM3:     XOR  A
92                    ;Maak (HL) tot einde regel
93 F344 77                        LD   (HL),A
94                    ;DE op begin nieuwe regel
95 F345 23                        INC  HL
96 F346 5D                        LD   E,L
97 F347 54                        LD   D,H
98                    ;HL op begin van de regel
99 F348 E1                        POP  HL
```

Hierin wordt de *verplaatste* BASIC regel onderzocht op een REM statement. Dit moet worden gedaan op de plaats waar een statement zal voorkomen. Dat is in elk geval direct na het regelnummer. HL moet daarvoor vier plaatsen omhoog. Met evenzoveel plaatsen moet de regellengte worden gecorrigeerd. Dit gebeurt in de LOOP van de regels 64 tot en met 67, nadat eerst de Accu met 4 is geladen. Steeds als de Accu met 1 is verminderd (regel 66) wordt nagegaan of deze al 0 is. Zolang dat niet het geval is wordt teruggegaan naar regel 64 waarna HL wordt opgehoogd en BC wordt verlaagd. Dit gaat door totdat de Accu 0 is. Dan is HL vier

plaatsen opgeschoven en geeft het adres van het eerste statement.
De volgende regels onderzoeken deze plaats of hier een REM-statement in is geladen. De token voor dit statement is 8F. Dat wordt in regel 69 in de Accu geladen. In regel 71 vindt de vergelijking plaats. Is (HL) inderdaad het token voor REM dan wordt met JR Z naar een volgend programmadeel gesprongen (ZKREM1). Is het statement niet dat voor REM dan kan het verderop nog in de regel voorkomen. Daarom moet de regel nu worden onderzocht op het scheidingsteken ":". Het token hiervoor (A3) wordt in de Accu geladen (regel 75). HL en BC bevatten de juiste gegevens voor CPIR. Wordt hierdoor geen scheidingsteken gevonden dan wordt door JR NZ naar het eind van dit programmadeel gesprongen voor het afsluiten van de regel. Wordt wel een scheidingsteken gevonden dan moet er opnieuw worden nagegaan of zich na dit teken een REM-statement bevindt. Daarvoor wordt teruggegaan naar regel 69 met JR Z (regel 80). Door CPIR heeft HL altijd een inhoud die 1 hoger is dan de laatst onderzochte geheugenplaats. HL geeft daarom weer de plaats van een statement aan. Dit gaat zo door totdat een REM-statement is gevonden en een sprong naar ZKREM1 wordt gemaakt, of tot het einde van de regel is bereikt en door regel 78 naar ZKREM3 wordt gegaan voor het afsluiten van de BASIC-regel.
Gaan we er vanuit dat werkelijk een REM-statement is gevonden dan komen we op regel 82 terecht. Nu zijn er twee mogelijkheden: of er was werkelijk het woord REM gebruikt in de BASIC-regel, of er was een apostrof gebruikt. In het eerste geval bevat de regel alleen de code 8F. In het tweede geval worden er drie codegetallen gebruikt: A3, 8F en E6. Achtereenvolgens zijn dat de tokens voor ":REM'". In het eerste geval moet de BASIC-regel worden afgebroken op de plaats van het REM-statement. In het tweede geval moet op de plaats van het ":" teken worden afgebroken, één plaats voor het REM-statement. In beide gevallen blijft het voorgaande deel van de regel bestaan. Is de BASIC-regel een volledige REM-regel dan blijven het adres van de volgende regel en het regelnummer in elk geval in de regel staan. Dit dient om te voorkomen dat een programma niet zal werken omdat een BASIC-regel is verwijderd, terwijl dat een sprongadres was uit een ander deel van het programma. Is een apostrof gebruikt dan wordt door regel 88 een geheugenplaats teruggegaan. Hierna moet HL nogmaals worden gecorrigeerd. Dit is nodig omdat in regel 83 HL was opgehoogd.
Het afsluiten van een BASIC regel houdt in dat op de laatste plaats (nu door HL aangegeven) een "0" wordt geladen. Hiervoor moet de Accu eerst nul zijn. Dit geschiedt eenvoudig door de functie XOR (regel 91). Als laatste van dit programmadeel wordt HL met 1 verhoogd zodat deze het adres aangeeft waar de volgende regel moet komen. Dit adres wordt overgenomen door DE. Met POP HL bevat HL weer het beginadres van de BASIC-regel. Op deze plaats moet het beginadres komen van de volgende regel.

```
100
101
102                         ;Corrigeer regeladres
103 F349 ED5303F6 CORRAD:        LD   (SAVADL),DE
104 F34D 3A03F6                  LD   A,(SAVADL)
105 F350 77                      LD   (HL),A
106 F351 23                      INC  HL
107 F352 3A04F6                  LD   A,(SAVADH)
108 F355 77                      LD   (HL),A
109                         ;HL aan het beginadres
110 F356 2B                      DEC  HL
111
```

Of er nu een REM-statement is verwijderd of niet, steeds moet aan het begin van de verplaatste BASIC-regel het adres worden geplaatst waar de volgende regel moet beginnen. Dit adres wordt aangegeven door DE, ook als geen REM-statement is verwijderd en zelfs als er in werkelijkheid geen verplaatsing heeft plaatsgevonden. Het beginadres van de BASIC-regel, dus daar waar het adres van de volgende regel moet worden geladen, bevindt zich in HL. Het overbrengen van het adres in DE verloopt via de geheugenplaatsen SAVADL en SAVADH en is eenvoudig in het programmadeel te volgen. Aan het eind van het programmadeel is de situatie weer als vanouds: HL bevat het beginadres van de verplaatste regel en DE het adres waar de volgende regel moet komen.

```
112
113                         ;Einde programma?
114 F357 2A01F6   PGEND:         LD   HL,(NRGNRL)
115 F35A 7E                      LD   A,(HL)
116 F35B A7                      AND  A
117 F35C C20AF3                  JP   NZ,BRGLL
118 F35F 23                      INC  HL
119 F360 7E                      LD   A,(HL)
120 F361 A7                      AND  A
121 F362 C20AF3                  JP   NZ,BRGLL
122                         ;Sluit programma af
123 F365 3E00                    LD   A,0
124 F367 12                      LD   (DE),A
125 F368 13                      INC  DE
126 F369 12                      LD   (DE),A
```

```
127                          ;Set de BASIC-vectoren
128 F36A 6B                         LD    L,E
129 F36B 62                         LD    H,D
130 F36C 23                         INC   HL
131                          ;Set VARTAB
132 F36D 22C2F6                     LD    (0F6C2H),HL
133                          ;Set ARYTAB
134 F370 110B00                     LD    DE,000BH
135 F373 19                         ADD   HL,DE
136 F374 22C4F6                     LD    (0F6C4H),HL
137                          ;Set STREND
138 F377 22C6F6                     LD    (0F6C6H),HL
139 F37A C9                         RET
140                                 END
```

Nu wordt het tijd om eens na te gaan of alle regels van het BASIC programma zijn doorlopen. In de geheugenplaatsen NRGNRL en NRGNRH bevindt zich nog steeds het adres van de niet verplaatste regel van het oorspronkelijke BASIC-programma, althans als er op dat adres nog een nieuwe regel begint. Het adres wordt in HL geladen (regel 114). Bevindt zich in deze geheugenplaats *geen* 0 dan is er inderdaad nog een nieuwe regel en gaan we terug naar het begin van het programma (BRGLL), waar het geheel opnieuw begint voor de volgende BASIC-regel. We gaan dit als volgt na. De inhoud van de geheugenplaats wordt met LD A,(HL) in de Accu geladen. Is deze 0 dan wordt door de AND bewerking de Z-FLAG geset. Is deze niet 0 dan verandert de inhoud door de AND-bewerking niet en de Z-FLAG wordt 0. Bevindt zich in deze geheugenplaats die door HL wordt aangewezen wel een 0 dan behoeft dat nog niet het eind van het programma te betekenen. Het is de lage byte van een adres en die kan ook 0 zijn! Daarom onderzoeken we ook de volgende geheugenplaats (INC HL) en als deze niet 0 is gaan we terug naar BRGLL. Over de inhoud van HL hoeven we ons geen zorgen te maken, die krijgt aan het begin van het programma weer de juiste waarde. Hebben we wel het eind van het programma gevonden dan moet de verplaatste regel nog afgesloten worden met twee keer 0. Dit vindt plaats in de regels 123 tot en met 126. Nu is deze laatste regel afgesloten met drie keer 0 zoals dat bij een BASIC-programma gebruikelijk is. Als laatste moeten nog drie vectoren worden gecorrigeerd. Ten eerste de vector die het eind van het BASIC-programma aangeeft en daarmee het begin van de variabeleruimte markeert, VARTAB. Eerst wordt de inhoud van DE overgenomen door HL en deze laatste met 1 verhoogd. HL bevat nu het beginadres van de variabeleruimte en zijn inhoud kan daarom in VARTAB worden geladen (regel 132). ARYTAB en STREND moeten van een adres worden voorzien dat 11 hoger is dan dat van VARTAB. Daarom wordt DE met 000B geladen en opgeteld bij HL (regel 135). Na het laden van HL in ARYTAB en STREND keren we weer terug naar BASIC met RET.

Als u het verloop van het programma hebt gevolgd dan hebt u niet alleen een voorbeeld van machinetaalprogrammeren gehad, maar u heeft dan ook nog eens kunnen zien op welke manier een BASIC-programma in de geheugenruimte is opgeslagen. Na het laden van een BASIC-programma kunt u het machinetaalprogramma vanuit de commandomode oproepen met

DEFUSR = &HF300: D = USR(0)

We kunnen echter ook een eigen commando daarvoor gebruiken. Als u verder niets anders dan uw cassettedeck op de computer hebt aangesloten dan is het commando KILL hiervoor wel geschikt. De HOOK H-KILL moet dan worden voorzien van JP F300. In het volgende programma vindt u de regels die daarvoor nodig zijn. U kunt er tevens het machinetaalprogramma mee invoeren, als u dat niet met een assemblerprogramma doet. RUN het programma en toets daarna KILL in. Negeer de tekst die op het scherm komt. Met LIST kunt u waarnemen dat inderdaad de gebruikte REM-statements uit het programma in het geheugen zijn verdwenen.

```
10 '****************
20 '* Invoeren van *
30 '*      REM     *
40 '*   destroyer  *
50 '****************
60 '
70 'Het veranderen van H-KILL
80 AD=&HFDFE: POKE AD,&HC3
90 POKE AD+1,0: POKE AD+2,&HF3
100 REM Invoeren van
110 REM machinetaalprogramma
120 FOR AD=&HF300 TO &HF37A: READ A$
130 POKE AD,VAL("&H"+A$): NEXT
140 DATA 2A,76,F6,22,01,F6,ED,5B,01
150 DATA F6,2A,01,F6,E5,D5,5E,23,56
160 DATA ED,53,01,F6,13,EB,A7,ED,52
170 DATA 2B,4D,44,D1,E1,D5,C5,ED,B0
180 DATA C1,E1,E5,3E,04,23,0B,3D,20
190 DATA FB,3E,8F,BE,28,08,3E,3A,ED
200 DATA B1,20,0A,2B,F3,3E,E6,23,BE
210 DATA 20,01,2B,2B,AF,77,23,5D,54
220 DATA E1,ED,53,03,F6,3A,03,F6,77
230 DATA 23,3A,04,F6,77,2B,2A,01,F6
240 DATA 7E,A7,C2,0A,F3,23,7E,A7,C2
250 DATA 0A,F3,3E,00,12,13,12,6B,62
260 DATA 23,22,C2,F6,11,0B,00,19,22
270 DATA C4,F6,22,C6,F6,C9:'*Einde*
```

8.3. Het uitschakelen van de CTRL-STOP toetsen.

U kent waarschijnlijk wel de mogelijkheid om in een BASIC- programma de CTRL-STOP toetsen uit te schakelen. De volgende BASIC- regels kunnen daarvoor worden gebruikt:

```
10 '
20 'Schakel stoptoets uit
30 '
40 ON STOP GOSUB 130
50 CLS: STOP ON
60 '
70 'Schakel stoptoets in bij A=20
80 '
90 IF A=20 THEN STOP OFF
100 PRINT CHR$(11)A: A=A+1
110 FOR X=0 TO 1000: NEXT
120 GOTO 90
130 RETURN
```

De werking wordt opgeroepen met de regels 40 en 50. Zodra de CTRL en STOP toetsen gelijktijdig worden ingedrukt dan wordt naar de subroutine gesprongen op regel 130, die alleen maar tot gevolg heeft dat weer gewoon wordt doorgegaan met het programma. Het programma heeft als resultaat dat in de linker bovenhoek van het scherm een nummer wordt geprint. Zodra dit nummer 20 is wordt in regel 90 de werking van de CRTL-STOP toetsen weer hervat. Dit is dus een methode die u kunt volgen als u niet wenst dat een bepaald gedeelte van het programma (of het gehele programma) kan worden onderbroken. Dezelfde werking kan worden verkregen met het volgende programma.

```
10 '
20 'Schakel stoptoets uit
30 '
40 CLS: A=0: POKE &HFBB1,1
50 '
60 'Schakel stoptoets in bij A=20
70 '
80 IF A=20 THEN POKE &HFBB1,0
90 PRINT CHR$(11)A: A=A+1
100 FOR X=0 TO 1000: NEXT
110 GOTO 80
```

Het geheim ligt bij de routine ISCNTC die kan worden aangeroepen via CALL 00BAH. Deze subroutine wordt in werking gesteld als de CTRL-STOP toetsen zijn gebruikt en onderzoekt onder andere de geheugenplaats BASROM op het adres FBB1. Vindt hij hierin de waarde 00H dan wordt de CTRL-STOP opdracht uitgevoerd. Vindt hij hierin een andere waarde dan 00H dan wordt de opdracht niet uitgevoerd en wordt het programma gewoon voortgezet.

9. In- en uitvoer van data

9.1. Karakters naar het beeldscherm.
Het ASCII-codesysteem is wel het meest toegepaste systeem bij het coderen van karakters voor bijvoorbeeld het opslaan en verwerken in datasystemen zoals computers. Ook de codegetallen die bij de MSX-karakterset zijn gebruikt zijn ontleend aan dit systeem. Er zijn echter ook een aantal grafische tekens in de MSX-karakterset die niet in het ASCII-codesysteem voorkomen. Deze hebben uiteraard wel een codegetal toegewezen gekregen. Lijst 24 toont u de volledige MSX-karakterset met de daarbij behorende codegetallen. Als u een toets op het toetsenbord indrukt dan heeft dat als resultaat dat het bijbehorende codegetal wordt opgeslagen in een of andere geheugenplaats. Dit is in eerste instantie een geheugenplaats van de *Keyboard Buffer* (KEYBUF, FBF0 tot en met FC18). Tijdelijk kan daarom het invoeren via het toetsenbord van de data sneller gaan dan het verwerken daarvan. Ook het toetsenbord moet door een programma worden "bewaakt". Dat wil zeggen dat voortdurend naar het toetsenbord moet worden gekeken of er niet een toets is ingedrukt. Hiervoor wordt door een timer een interrupt veroorzaakt. Dit gebeurt vijftig keer per seconde. De MSX-computer werkt in de INT-mode 1 (paragraaf 5.5.), zodat elke 20 milliseconden het lopende programma wordt onderbroken en naar de interrupt-routine wordt gesprongen op het adres 0038. Deze routine (KEYINT) heeft als doel om diverse invoerapparaten (joy-sticks, toetsenbord) te controleren maar heeft ook nog ander functies (het verhogen van JIFFY-clock bijv.). Het is deze routine die het toetsenbord *"SCAND"*, als het ware "afzoekt" of er een toets is ingedrukt. Dit heeft elke 20 milliseconden plaats en dat is snel genoeg. U kunt een toets niet zo kort indrukken of het is wel langer dan 20 milliseconden. Is er een toets ingedrukt dan wordt het codegetal van het desbetreffende karakter in de keyboard-buffer geladen. Met een andere routine (CHGET op adres 009F) kan worden nagegaan of er een karakter in de keyboard-buffer is opgeslagen. Daarna leest dezelfde routine een karakter uit de buffer en laadt het codegetal daarvan in de Accu. De keyboard-buffer is een "last in - last out" geheugen, dat wil zeggen dat het karakter dat het laatst is ingetoetst ook het laatst wordt uitgelezen. De INT is maskeerbaar met een DI- instructie. Dat betekent dat na een DI-instructie het toetsenbord niet meer te gebruiken is omdat de interrupt niet meer optreedt. Ook de Jyffy-clock staat dan stil. Dit is bijvoorbeeld het geval tijdens het saven van data naar de tape.
Met het volgende programma kunnen we karakters intoetsen en zichtbaar maken op het beeldscherm.

```
1                          ;*************
2                          ;* Karakters *
3                          ;*  op het   *
4                          ;*  scherm   *
5                          ;*************
6
7
8                                 ORG  0E000H
9                                 LOAD 0E000H
10
11                         ;Wis het scherm
12 E000 CDC300             CLS:     CALL 00C3H
13                         ;Test de CTRL-STOP toets
14 E003 CDBA00             ISCNTC:  CALL 00BAH
15                         ;Invoeren van het karakter
16 E006 CD9F00             CHGET:   CALL 009FH
17                         ;Is CR ingevoerd?
18 E009 FE0D                        CP   0DH
19 E00B 2813                        JR   Z,CR
20                         ;Karakter op het scherm
21 E00D CDA200             CHPUT:   CALL 00A2H
22                         ;Is BS ingevoerd?
23 E010 FE08                        CP   08H
24 E012 20EF                        JR   NZ,ISCNTC
25                         ;Bij BS laatste kar. verw.
26 E014 3E20               SPATIE:  LD   A,20H
27 E016 CDA200                       CALL 00A2H
28                         ;Een plaats terug (BS)
29 E019 3E08                        LD   A,08H
30 E01B CDA200                      CALL 00A2H
31 E01E 18E3                        JR   ISCNTC
32                         ;Bij CR, nieuwe regel (LF)
33 E020 3E0A               CR:      LD   A,0AH
34 E022 CDA200                      CALL 00A2H
35 E025 3E0D                        LD   A,0DH
36 E027 CDA200                      CALL 00A2H
37 E02A C9                          RET
38                                  END
```

Er zijn in dit programma diverse routines uit de BASIC ROM gebruikt. Om te beginnen op regel 12 de routine CLS. Door deze simpelweg aan te roepen maken we het scherm schoon en plaatsen we de cursor in de linkerbovenhoek van het scherm. Verder is het ook wel handig om met de CTRL-STOP-toetsen het programma weer te kunnen verlaten.

Dit is mogelijk door de routine ISCNTC aan te roepen (regel 14). Deze routine gaat na of de CTRL-STOP-toetsen zijn ingedrukt. Is dat het geval dan wordt de computer in de commandomode geplaatst. Het programma keert aan het eind weer terug bij regel 14 zodat steeds de CTRL-STOP-toetsen worden gecontroleerd. Op regel 16 volgt de CHGET-routine. Deze routine leest een karakter uit de keyboard-buffer, als deze daarin tenminste aanwezig is. Is de keyboard-buffer leeg dan wacht deze routine tot een karakter in de buffer wordt geladen, dus in principe tot een toets wordt ingedrukt. Is dat gebeurd dan wordt eerst nagegaan of de ingedrukte toets die van Cariage Return (CR) is. In dat geval is het laatste karakter ingevoerd en moet het programma worden verlaten. Er wordt

Hexa-decimale code	00–1F		20–3F		40–5F		60–7F		80–9F		A0–BF		C0–DF		E0–FF	
	code	teken	code	teken	code	teken	code	teken	code	teken	code	teken	code	teken	code	teken
0	0	(nul)	32	(spatie)	64	@	96	`	128	Ç	160	á	192	▬	224	α
1	1	☺	33	!	65	A	97	a	129	ü	161	í	193	▚	225	β
2	2	☻	34	"	66	B	98	b	130	é	162	ó	194	▌	226	Γ
3	3	♥	35	#	67	C	99	c	131	â	163	ú	195	▬	227	Π
4	4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	▪	228	Σ
5	5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	█	229	σ
6	6	♠	38	&	70	F	102	f	134	å	166	ª	198	▏	230	μ
7	7	•	39	'	71	G	103	g	135	ç	167	º	199	▞	231	γ
8	8	◘	40	(	72	H	104	h	136	ê	168	¿	200	▌	232	Φ
9	9	○	41	)	73	I	105	i	137	ë	169	⌐	201	▐	233	Θ
A	10	◙	42	*	74	J	106	j	138	è	170	¬	202	▌	234	Ω
B	11	♂	43	+	75	K	107	k	139	ï	171	½	203	⁄⁄	235	δ
C	12	♀	44	,	76	L	108	l	140	î	172	¼	204	⧵⧵	236	∞
D	13	♪	45	—	77	M	109	m	141	ì	173	¡	205	▼	237	∅
E	14	♫	46	.	78	N	110	n	142	Ä	174	«	206	▲	238	∈
F	15	☼	47	/	79	O	111	o	143	Å	175	»	207	▶	239	∩
0	16	┼	48	Ø	80	P	112	p	144	É	176	Ã	208	◀	240	≡
1	17	┴	49	1	81	Q	113	q	145	æ	177	ã	209	⧗	241	±
2	18	┬	50	2	82	R	114	r	146	Æ	178	Ĩ	210	⧓	242	≥
3	19	┤	51	3	83	S	115	s	147	ô	179	ĩ	211	▖	243	≤
4	20	├	52	4	84	T	116	t	148	ö	180	Õ	212	▗	244	⌠
5	21	┼	53	5	85	U	117	u	149	ò	181	õ	213	▪	245	⌡
6	22	│	54	6	86	V	118	v	150	û	182	Ũ	214	▪	246	÷
7	23	─	55	7	87	W	119	w	151	ù	183	ũ	215	✲	247	≈
8	24	┌	56	8	88	X	120	x	152	ÿ	184	Ĳ	216	△	248	°
9	25	┐	57	9	89	Y	121	y	153	Ö	185	ĳ	217	‡	249	•
A	26	└	58	:	90	Z	122	z	154	Ü	186	¾	218	ω	250	‐
B	27	┘	59	;	91	[	123	{	155	¢	187	∽	219	█	251	√
C	28	╳	60	<	92	\	124	¦	156	£	188	◇	220	▄	252	η
D	29	╱	61	=	93	]	125	}	157	¥	189	‰	221	▌	253	²
E	30	╲	62	>	94	∧	126	~	158	₧	190	¶	222	▐	254	■
F	31	┼	63	?	95	_	127		159	ƒ	191	§	223	▀	255	

Lijst 24. Codetekens van de MSX-karakterset.

daarvoor een sprong gemaakt naar regel 33. Is een karakter ingevoerd dan wordt dit op het scherm getoond. Door CHGET is het codegetal van het karakter in de Accu geladen. Door nu de routine CHPUT aan te roepen op het adres 00A2 wordt dit karakter op het scherm getoond. Het kan natuurlijk zijn dat u een verkeerd karakter hebt ingevoerd. Als u dat wilt herstellen dan kan de toets Back Space (BS) worden gebruikt. De code voor deze toets is 08H. In regel 23 wordt nagegaan of deze toets is gebruikt. De routine CHPUT verandert het getal in de Accu namelijk niet. Is deze toets niet gebruikt dan wordt voor het volgende karakter teruggegaan naar ISCNTC. Is de toets BS wel gebruikt dan is in regel 21 de cursor op het scherm al een plaats teruggegaan. Deze plaats moet eerst nog ''leeg'' worden. Dat kan door een spatie (codegetal 20H) op deze plaats te schrijven. Hiervoor zijn de regels 26 en 27. Nu is de cursor weer een plaats opgeschoven en moet daarom weer terug. In de regels 29 en 30 wordt dat tot stand gebracht, waarna weer naar ISCNTC wordt gesprongen. Na het gebruiken van de CR toets komen we op regel 33 terecht. Het codeteken voor CR (0DH) heeft slechts tot gevolg dat naar het begin van de regel wordt gegaan. Er moet echter naar het begin van de volgende regel worden gesprongen. Daarom wordt eerst het codeteken voor ''nieuwe regel'', Line Feet (LF), in de Accu geladen (0AH) en dan de routine CHPUT aangeroepen. Hierna vindt hetzelfde plaats maar nu voor het codeteken 0DH. Uiteindelijk wordt met RET het programma verlaten.

9.2. Karakters naar de printer.

Op gelijke wijze kunnen ook karakters naar de printer worden gestuurd.

```
1                    ;*************
2                    ;* Karakters *
3                    ;*  naar de  *
4                    ;*  printer  *
5                    ;*************
6
7
8                               ORG  0E000H
9                               LOAD 0E000H
10
11                   ;Verw. functietoetsdisplay
12 E000 CDCC00       ERAFNK:    CALL 00CCH
13                   ;Wis het scherm
14 E003 AF                      XOR  A
15 E004 CDC300       CLS:       CALL 00C3H
16                   ;Test de CTRL-STOP toets
17 E007 CDBA00       ISCNTC:    CALL 00BAH
18                   ;Invoeren van het karakter
19 E00A CD9F00       CHGET:     CALL 009FH
20                   ;Is CR ingevoerd?
21 E00D FE0D                    CP   0DH
22 E00F 2808                    JR   Z,CR
23                   ;Karakter op het scherm
24 E011 CDA200       CHPUT:     CALL 00A2H
25                   ;Karakter naar printer
26 E014 CDA500       LPTOUT:    CALL 00A5H
27 E017 18EE                     JR   ISCNTC
28                   ;Bij CR, nieuwe regel (LF)
29 E019 3E0A         CR:        LD   A,0AH
30 E01B CDA200                  CALL 00A2H
31 E01E CDA500                  CALL 00A5H
32 E021 3E0D                    LD   A,0DH
33 E023 CDA200                  CALL 00A2H
34 E026 CDA500                  CALL 00A5H
35                   ;Toon functietoetsdisplay
36 E029 CDCF00       DSPFNK:    CALL 00CFH
37 E02C C9                      RET
38                              END
```

De opbouw van dit programma is ongeveer gelijk aan dat van het voorgaande. De eerste nieuwigheid zit in regel 12. Hier is de routine ERAFNK in aangeroepen op het adres 00CC. Het resultaat hiervan is dat de functies van de functietoetsen niet meer onderaan het scherm zichtbaar zijn. Een ander resultaat is dat door de routine de Z-FLAG is gereset. Het gevolg hiervan is weer dat de routine CLS niet meer werkt en het scherm niet meer schoonmaakt. Alvorens CLS aan te roepen maken we Z eerst 0 met een bewerking die de inhoud van de Accu nul maakt: XOR A (regel 14). Tot en met regel 24 is er niets nieuws onder de zon. Om op het beeldscherm te kunnen zien wat we intoetsen gebruiken we de routine CHPUT. De code van het ingetoetste karakter bevindt zich nu nog steeds in de Accu en dat is nodig als de routine LPTOUT in regel 26 op het adres 00A5 wordt aangeroepen. Deze routine stuurt het karakter naar de printer. Is de printer niet gereed om het karakter te ontvangen dan blijft de routine wachten tot de printer meldt dat hij gereed is. Het karakter wordt in de buffer van de printer geladen. Voor het volgende karakter gaan we terug naar ISCNTC. De printer zal in eerste instantie niet reageren. Zijn echter 80 karakters ontvangen dan worden deze op het papier afgedrukt. Moeten minder dan tachtig karakters op een regel worden afgedrukt dan is een Cariage Return nodig. Direct na het ontvangen hiervan gaat de printer aan het werk. Ook nu moet eerst een Line Feet worden verzonden. Dit heeft plaats in de regels 29 tot en met 34. Als laatste onderdeel van het programma moeten de functies van de functietoetsen weer op het scherm verschijnen. Hiervoor zorgt de routine DSPFNK op het adres 00CF.

9.3. Karakters in het geheugen.

Karakters worden steeds in de vorm van een string in het geheugen geplaatst. Voor het invoeren van de karakters vanaf het toetsenbord kan weer de routine CHGET op het adres 009F worden gebruikt. De ingevoerde karakters worden in de volgorde van het invoeren na elkaar in het geheugen geladen. Hiervoor is een pointer nodig die steeds het adres aangeeft waar het volgende karakter moet worden ingeschreven. In het volgende programma is hiervoor het registerpaar HL gebruikt.

```
 1                ;*************
 2                ;* Karakters *
 3                ;*  in het   *
 4                ;* geheugen  *
 5                ;*************
 6
 7
 8                           ORG   0E000H
 9                           LOAD  0E000H
10
11                ;Variabele adres in HL
12 E000 2100E8               LD    HL,0E800H
13                ;Wis het scherm
14 E003 CDC300               CALL  00C3H
15                ;Test de CTRL-STOP toets
16 E006 CDBA00    ISCNTC:    CALL  00BAH
17                ;Invoeren van het karakter
18 E009 CD9F00    CHGET:     CALL  009FH
19                ;Is CR ingevoerd?
20 E00C FE0D                 CP    0DH
21 E00E 2818                 JR    Z,CR
22                ;Karakter op het scherm
23 E010 CDA200    CHPUT:     CALL  00A2H
24                ;Is BS ingevoerd?
25 E013 FE08                 CP    08H
26 E015 2804                 JR    Z,BCKSPC
27                ;Karakter in het geheugen
28 E017 77                   LD    (HL),A
29                ;Volgende geheugenplaats
30 E018 23                   INC   HL
31                ;Voor volgend karakter
32 E019 18EB                 JR    ISCNTC
33                ;Bij BS laatste kar. verw.
34 E01B 3E20      BCKSPC:    LD    A,20H
35 E01D CDA200               CALL  00A2H
36                ;Een plaats terug (BS)
37 E020 3E08                 LD    A,08H
38 E022 CDA200               CALL  00A2H
39                ;Een geheugenplaats terug
40 E025 2B                   DEC   HL
41                ;Voor volgend karakter
42 E026 18DE                 JR    ISCNTC
43                ;Bij CR, nieuwe regel (LF)
44 E028 3E0A      CR:        LD    A,0AH
45 E02A CDA200               CALL  00A2H
46 E02D 3E0D                 LD    A,0DH
47 E02F CDA200               CALL  00A2H
48                ;String afsluiten met "0"
49 E032 3600                 LD    (HL),0
50 E034 C9                   RET
51                           END
```

In regel 12 wordt HL met het adres voor het eerste karakter geladen. Belangrijk bij het intoetsen is dat we steeds kunnen zien wat we doen. Daarom worden de ingevoerde karakters ook op het scherm zichtbaar gemaakt. Dit scherm wordt eerst in regel 14 schoongemaakt. Ook bij dit programma is het nuttig om de CNTR-STOP-toetsen te kunnen gebruiken om het te onderbreken (regel 16). De regels 18 tot en met 23 zijn we al eens eerder tegengekomen. Hierna wordt nagegaan of een BS is ingevoerd. Is dat het geval dan wordt een aantal regels overgeslagen. In deze regels wordt het karakter in het geheugen geladen. De ASCII-code hiervoor is door CHGET in de Accu terechtgekomen. Nu moet de pointer worden verhoogd zodat deze het adres voor het volgende karakter aangeeft. Om dit karakter te kunnen intoetsen moet worden teruggegaan naar het begin van het programma. Is de BS-toets gebruikt dan wordt eerst het laatst ingetoetste karakter verwijderd en daarna weer een plaats teruggegaan op het scherm. Ook in het variabelegeheugen moet een plaats worden teruggegaan. Hiervoor wordt de pointer in HL met 1 verlaagd. Voor het intoetsen van het volgende karakter wordt weer teruggegaan naar ISCNTC. Als de string is ingevoerd moet aan het eind een RETURN worden gegeven. Voor het beeldscherm moet de CR worden voorafgegaan door een LF. In het geheugen moet het einde van de string worden gemarkeerd. We doen dit hier door na het laatste karakter een "0" in het geheugen te plaatsen (regel 49). Voer het programma in de computer en start het. Toets nu in:

MSX machinetaal.

Het resultaat kunt u vaststellen met:

```
FOR X=0 TO 18: PRINT
CHR$(PEEK(&HE800+X));:NEXT
```

9.4. Het printen van een string.

Voor het printen van een string, die bijvoorbeeld met het voorgaande programma in het geheugen is gebracht, worden de karakters stuk voor stuk uit het geheugen gelezen en op het scherm geschreven. Omdat het einde van de string is gemarkeerd met een "0" is de totale lengte van de string eenvoudig vast te stellen. Het uitlezen van de karakters kan worden voortgezet tot deze "0" wordt gevonden. Een nieuwe routine in dit programma is POSIT op het adres 00C6. Met deze routine kan vooraf de plaats van de cursor worden toegewezen.

```
1                      ;***************
2                      ;*   String    *
3                      ;*   op het    *
4                      ;* beeldscherm *
5                      ;***************
6
7                               ORG  0E000H
8                               LOAD 0E000H
9
10
11                     ;Wis het scherm
12  E000 CDC300                 CALL 00C3H
13                     ;Geef de CSR zijn plaats
14                     ;CSR op kolomnr. 10
15  E003 260A                   LD   H,0AH
16                     ;CSR op regelnr. 12
17  E005 2E0C                   LD   L,0CH
18  E007 CDC600        POSIT:   CALL 00C6H
19                     ;Beginadres string in HL
20  E00A 2100E8                 LD   HL,0E800H
21                     ;Haal karakter uit geh.
22  E00D 7E            PRNTCH:  LD   A,(HL)
23                     ;Verhoog pointer
24  E00E 23                     INC  HL
25                     ;Einde van de string?
26  E00F A7                     AND  A
27  E010 2805                   JR   Z,LFCR
28                     ;Karakter op het scherm
29  E012 CDA200        CHPUT:   CALL 00A2H
30                     ;Voor volgend karakter
31  E015 18F6                   JR   PRNTCH
32                     ;Geef LF en CR
33  E017 3E0A          LFCR:    LD   A,0AH
34  E019 CDA200                 CALL 00A2H
35  E01C 3E0D                   LD   A,0DH
36  E01E CDA200                 CALL 00A2H
37  E021 C9                     RET
38                              END
```

Nadat het scherm is schoongemaakt wordt de cursor op zijn plaats gebracht. Hiervoor moet het nummer van de kolom (het plaatsnummer vanaf de linkerkant van het scherm) worden ingevoerd in het H- register. Het regelnummer (plaatsnummer vanaf de bovenkant van het scherm) moet in het L-register worden geschreven. Door nu de routine POSIT op het adres 00C6 aan te roepen (regel 18) wordt de cursor op de gewenste plaats van het scherm gebracht. Voor het uitlezen van de karakters is weer een pointer nodig. Ook nu gebruiken we daarvoor het HL-registerpaar. Om te beginnen wordt dit met het adres van het eerste karakter van de string geladen (regel 20). In regel 22 wordt dit karakter in de Accu geladen. In de volgende regel wordt de pointer opgehoogd en geeft nu de plaats van het volgende karakter aan. Het einde van de string is bereikt als de gelezen geheugenplaats een 0 zou bevatten. Door de AND A bewerking in regel 26 verandert de inhoud van de Accu niet. Is de Accu echter leeg dan wordt de Z-FLAG geset en door regel 27 naar het eind van het programma gesprongen. Zolang dat nog niet het geval is wordt in regel 29 het karakter op het scherm geprint. Daarna wordt weer teruggegaan naar het begin van het programma (PRNTCH). Is het einde van de string bereikt dan worden op de bekende manier een LF en een CR naar het scherm gestuurd.

Met een overeenkomend programma kunnen ook de karakters van een string naar de printer worden verzonden. Uiteraard kan dan het plaatsen van de cursor achterwege blijven. Verder moet de instructie CALL 00A2 op de regels 29, 34 en 36 worden vervangen door CALL 00A5 voor de routine LPTOUT.

9.5. Werken met het toetsenbord.

De toetsen van het toetsenbord zijn in principe niet anders dan schakelaars die worden gesloten als de bijbehorende toetsen worden ingedrukt. Om te kunnen "uitlezen" welke toets is ingedrukt zijn ze in een *"matrix"* opgenomen. Dit systeem kan voorgesteld worden als een aantal elkaar kruisende geleiders die geen contact met elkaar maken, maar waarbij de "knooppunten" kunnen worden doorverbonden door het indrukken van een schakelaar. Bij de toetsenbordmatrix zijn negen horizontale geleiders, de regels, en acht verticale geleiders, de kolommen (figuur 68). Door middel van een elektronische schakeling kan één van de negen regels actief worden gemaakt. De regels zijn in rust

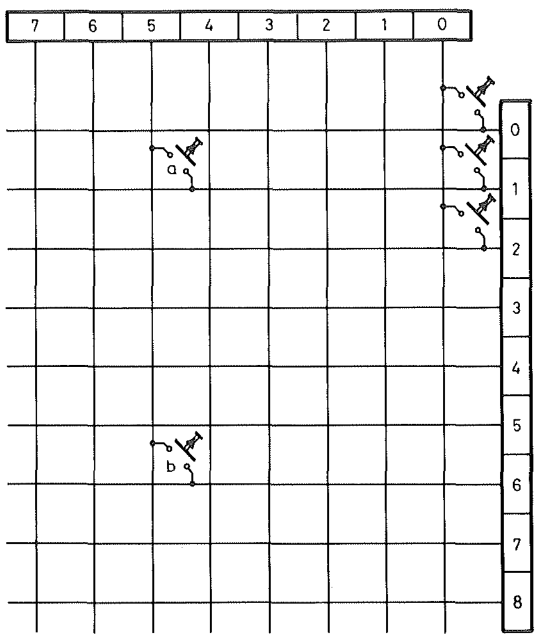


Fig. 68. De toetsenbordmatrix.

''hoog'' (te beschouwen als ''1'') maar actief ''laag'' (0). Door een getal van 0 tot en met 8 naar een uitgangspoort te schrijven wordt de overeenkomstige regel actief (0). Een ingangspoortregister is aangesloten op de acht kolommen. De acht poorten hiervan zijn ''hoog'' als alle schakelaars geopend zijn. Stel dat de schakelaar a van figuur 68 gesloten is. Als regel 6 actief is zal het getal 11111111 uit het ingangspoortregister kunnen worden gelezen. Is echter regel 1 actief dan wordt het getal 11011111 uit het register gelezen. Een bepaalde schakelaar (toets) wordt daarom gemarkeerd door een combinatie van een regelnummer en het getal dat uit het ingangspoortregister kan worden gelezen. Nemen we het regelnummer als hoge byte en het getal in het ingangspoortregister als lage byte dan is de code voor toets a:
$0000000111011111_{bin} = 01DF_{hex}$. Met deze acht bij negen matrix kunnen precies alle 72 toetsen van het MSX-toetsenbord worden aangesloten. Een opsomming van de codegetallen vindt u in lijst 25.

Toets	Code	Toets	Code	Toets	code
0	00FE	C	03FE	SHIFT	06FE
1	00FD	D	03FD	CTRL	06FD
2	00FB	E	03FB	GRAPH	06FB
3	00F7	F	03F7	CAPS L	06F7
4	00EF	G	03EF	CODE	06EF
5	00DF	H	03DF	F1	06DF
6	00BF	I	03BF	F2	06BF
7	007F	J	037F	F3	067F
8	01FE	K	04FE	F4	07FE
9	01FD	L	04FD	F5	07FD
-	01FB	M	04FB	ESC	07FB
=	01F7	N	04F7	TAB	07F7
\	01EF	O	04EF	STOP	07EF
[	01DF	P	04DF	BS	07DF
]	01BF	Q	04BF	SELECT	07BF
;	017F	R	047F	RETURN	077F
'	02FE	S	05FE	SPATIE	08FE
~	02FD	T	05FD	CLS	08FD
,	02FB	U	05FB	INS	08FB
/	02EF	W	05EF	CRSR L	08EF
BLANC	02DF	X	05DF	CRSR U	08DF
A	02BF	Y	05BF	CRSR D	08BF
B	027F	Z	057F	CRSR R	087F

Lijst 25. De codegetallen van de toetsen.

Om te bepalen welke toets is ingedrukt wordt een techniek gebruikt die *''SCANNEN''* wordt genoemd. Een voorbeeld hiervan vindt u in het volgende programma.

```
1                        ;***************
2                        ;*   Scannen   *
3                        ;*   van het   *
4                        ;* toetsenbord *
5                        ;***************
6
7
8                               ORG   0E000H
9                               LOAD  0E000H
10
11
12                       ;Vertraag
13 E000 2E00                    LD    L,0
14 E002 65                      LD    H,L
15 E003 55                      LD    D,L
16 E004 5D                      LD    E,L
17 E005 1B              VRTRG:  DEC   DE
18 E006 CD2000          DCOMPR: CALL  0020H
19 E009 20FA                    JR    NZ,VRTRG
20                       ;Voor negen regels
21 E00B 2609            SCAN:   LD    H,09H
22 E00D 7C              SCAN1:  LD    A,H
23 E00E 3D                      DEC   A
24                       ;Laatste regel?
25 E00F FA0BE0                  JP    M,SCAN
26 E012 67                      LD    H,A
27 E013 CD4101          SNSMAT: CALL  0141H
28 E016 FEFF                    CP    0FFH
29 E018 28F3                    JR    Z,SCAN1
30 E01A 6F                      LD    L,A
31 E01B 2200E8                  LD    (0E800H),HL
32 E01E C9                      RET
33                              END
```

Op welke manier u het programma invoert en op welke manier u het daarna start, steeds zult u als laatste toets de RETURN-toets hebben gebruikt. Nu is de computer akelig snel en nog voordat u de toets hebt kunnen loslaten wordt het programma al door de computer afgewerkt. Om te voorkomen dat de nog ingedrukte RETURN- toets u parten speelt is een vertraging in het programma ingebouwd. Hiervoor zijn de regels 13 tot en met 19. Eerst worden de registerparen HL en DE 0 gemaakt. Direct daarna wordt DE met 1 verlaagd. Deze heeft nu FFFF als inhoud. In de BASIC-ROM kunnen we de subroutine ''DCOMPR'' op het adres 0020 aanroepen. Deze vergelijkt de inhouden van HL en DE (regel 18). Zijn deze gelijk dan wordt de Z-FLAG geset. Dat is voorlopig nog niet het geval en in een lus wordt DE steeds opnieuw verlaagd tot zijn inhoud 0 is. De lus moet daarvoor wel 65536 keer worden doorlopen en dat kost de nodige tijd. In deze tijd kunt u de toets allang weer hebben losgelaten. De grootte van de vertraging is eenvoudig te regelen door het getal in DE een andere beginwaarde te geven. Hierna volgt het eigenlijke scannen van het toetsenbord. Voor het inschrijven en het uitlezen van de poortregisters voor het toetsenbord maken we gebruik van een routine in de BASIC-ROM, de routine

"SNSMAT" op het adres 0141. Het gekozen regelnummer wordt eerst in de Accu geladen en daarna wordt de routine aangeroepen. Als resultaat geeft dan de routine de waarde van de acht kolommen weer in de Accu. Om te beginnen wordt de Accu met het nummer 9 geladen. Deze regel bestaat niet en de Accu wordt dan ook direct daarop met 1 verlaagd (regels 21 tot en met 23). Nu wordt nagegaan of de laatste regel van de matrix al is gecontroleerd. We gaan van het hoogste regelnummer (8) naar het laagste (0) en als na het laagste regelnummer de Accu wordt verlaagd dan wordt zijn inhoud negatief. Het regelnummer wordt in het H-register bewaard voordat de routine SNSMAT wordt aangeroepen. Is op de desbetreffende regel geen toets ingedrukt dan zal de vergelijking van de inhoud van de Accu met FF een sprong naar SCAN1 opleveren. Hier zal de Accu weer van het oude regelnummer worden voorzien. Het nieuwe regelnummer zal 1 lager zijn (DEC A) en de volgende matrixregel wordt gecontroleerd. Is geen toets ingedrukt dan gaat de werking zo verder tot regel 0 is gecontroleerd. Daarna wordt door regel 25 weer teruggegaan naar het begin van dit programmadeel en het scannen herhaalt zich. Zolang er geen toets is ingedrukt zullen we dit programmadeel dus niet kunnen verlaten. Het heeft dan ook de functie van "wachten op het indrukken van een toets". Is eenmaal een toets ingedrukt dan wordt dat vastgesteld in de regels 28 en 29. Na afloop zal het registerpaar HL het codenummer van de toets bevatten. Dit wordt in de geheugenplaatsen E800 (lage byte) en E801 hoge byte geladen. Na het doorlopen van het programma kunt u deze eventueel vanuit BASIC uitlezen.

Behalve het scannen van het toetsenbord moet het codegetal van de desbetreffende toets worden omgezet tot een ASCII-getal voor het desbetreffende karakter. Er moet daarbij rekening worden gehouden dat ook combinaties mogelijk zijn (SHIFT, GRAPH, CODE). Het ASCII-getal moet daarna in de toetsenbordbuffer (KEYBUF) worden geladen. Deze handelingen worden elke 20 ms. verricht door de subroutine KEYINT van de BASIC-ROM. Als we zelf wat met de toetsen willen gaan doen is het vaak nodig om de werking van KEYINT uit te schakelen. Nu kan dat met de instructie DI. Vaak hebben de subroutines die we uit de BASIC-ROM gebruiken aan het eind de instructie EI staan. Dat houdt in dat de interrupts die we met DI onmogelijk hebben gemaakt, door de subroutine weer mogelijk worden en daar schieten we dan niet te veel mee op. Het is echter heel goed mogelijk de routine KEYINT definitief uit te schakelen. Direct aan het begin van deze routine vinden we de instructies

```
PUSH HL
PUSH DE
PUSH BC
PUSH AF
EXX
EX AF,AF
PUSH HL
PUSH DE
PUSH BC
PUSH AF
PUSH IY
PUSH IX
CALL 0FD9AH
```

Dat betekent dat alle werkregisters van de processor naar de STACK worden geschreven. Dat is nodig omdat de routine op elk moment kan worden aangeroepen en niet bekend is welke registers data bevatten die niet verloren mag gaan. Na deze handeling wordt de hoek H-KEYI aangeroepen. Deze hoek gaan we gebruiken om de routine uit te schakelen. In het volgende programma is daarvoor een apart deel opgenomen.

```
 1                 ;*************
 2                 ;* Selecteer *
 3                 ;*    een    *
 4                 ;*   toets   *
 5                 ;*************
 6
 7
 8                                ORG  OEOOOH
 9                                LOAD OEOOOH
10
11                 ;JP SCRINT op H-KEYI
12 E000 213FE0     DISINT:        LD   HL,SCRINT
13 E003 229BFD                    LD   (OFD9BH),HL
14 E006 3EC3                      LD   A,OC3H
15 E008 329AFD                    LD   (OFD9AH),A
16                 ;Wacht op toets los
17 E00B 2609       WCHTTL:        LD   H,09H
18 E00D 7C         WACHT1:        LD   A,H
19 E00E 3D                        DEC  A
20                 ;Geen toets in, naar SCAN
21 E00F FA21E0                    JP   M,SCAN
22 E012 67                        LD   H,A
23 E013 CD4101                    CALL 0141H
24 E016 FEFF                      CP   OFFH
25 E018 28F3                      JR   Z,WACHT1
26                 ;Toets nog in, opnieuw
27 E01A 18EF                      JR   WCHTTL
28                 ;Test de CTRL-STOP toetsen
29 E01C CDB700     BREAKX:        CALL 00B7H
30 E01F 3818                      JR   C,ENAINT
```

```
31                      ;Wacht op toets in
32 E021 11DF07          SCAN:       LD   DE,07DFH
33 E024 2609            SCAN1:      LD   H,09H
34 E026 7C              SCAN2:      LD   A,H
35 E027 3D                          DEC  A
36                      ;Geen toets in, opnieuw
37 E028 FA1CE0                      JP   M,BREAKX
38 E02B 67                          LD   H,A
39 E02C CD4101                      CALL 0141H
40 E02F FEFF                        CP   0FFH
41 E031 28F3                        JR   Z,SCAN2
42 E033 6F                          LD   L,A
43                      ;Vergelijk HL en DE
44 E034 CD2000          DCOMPR:     CALL 0020H
45                      ;Niet de goede toets, terug
46 E037 20E3                        JR   NZ,BREAKX
47                      ;Goede toets, C9 in H-KEYI
48 E039 3EC9            ENAINT:     LD   A,0C9H
49 E03B 329AFD                      LD   (0FD9AH),A
50 E03E C9                          RET
51
52
53                      ;Corrigeer CALL H-KEYI
54 E03F E1              SCRINT:     POP  HL
55                      ;Registers terug uit STACK
56 E040 DDE1                        POP  IX
57 E042 FDE1                        POP  IY
58 E044 F1                          POP  AF
59 E045 C1                          POP  BC
60 E046 D1                          POP  DE
61 E047 E1                          POP  HL
62                      ;Verwissel met altern. reg.
63 E048 08                          EX   AF,AF
64 E049 D9                          EXX
65 E04A F1                          POP  AF
66 E04B C1                          POP  BC
67 E04C D1                          POP  DE
68 E04D E1                          POP  HL
69                      ;Return van interrupt
70 E04E ED4D                        RETI
71                                  END
```

Het is een programma waarmee een toets wordt geselecteerd, dat wil zeggen dat gewacht wordt tot een bepaalde toets is ingedrukt. In de regels 12 tot en met 15 wordt H-KEYI veranderd tot JP SCRINT. Hierin is SCRINT het programmadeel dat de routine KEYINT uitschakelt. LD HL,SCRINT in regel 12 heeft tot gevolg dat het beginadres van dit programmadeel (E03F) in HL wordt geladen. Eerst wordt dit adres in FD9B en FD9C geschreven. Daarna pas wordt de hoek op FD9A van C9 veranderd in C3 (RET wordt veranderd in JP) Deze volgorde is nodig omdat anders, net na het veranderen van de hoek en voor het laden van het adres een interrupt zou kunnen optreden. Het aanroepen van H-KEYI door de routine zou dan tot gevolg hebben dat naar het adres C9C9 zou worden gesprongen, waardoor de computer op TILT slaat.

Als nu een interrupt optreedt dan wordt het programmadeel op regel 54 aangeroepen. Eerst wordt hierin de Stackpointer gecorrigeerd voor de CALL naar H-KEYI. We laden daarbij het retouradres, waarnaar gegaan moet worden om de routine KEYINT normaal verder te doorlopen, in HL. Dat heeft overigens geen enkele betekenis. Het heeft slechts tot doel de Stackpointer te corrigeren. Daarna wordt aan alle processorregisters weer de oorspronkelijke inhoud gegeven. Als laatste nemen we met RETI afscheid van de interruptroutine en keren we weer terug naar het adres waar het programma bij het optreden van de interrupt verlaten was. Het optreden van een interrupt heeft nu slechts tot gevolg dat de processorregisters van en naar de STACK worden geladen. Verder gebeurt er niets.

Na het veranderen van H-KEYI vervolgt het programma met het scannen van het toetsenbord in de regels 17 tot en met 25. Als een toets is ingedrukt wordt door regel 27 opnieuw naar het begin van dit programmadeel gesprongen en herhaalt het scannen zich. Inplaats van een bepaalde wachttijd zoals in het vorige programma, blijven we nu gewoon wachten tot de desbetreffende toets wordt losgelaten. Voor dit programmadeel is het noodzakelijk dat KEYINT wordt uitgeschakeld. We kunnen het beschrijven als het wachten op het loslaten van een ingedrukte toets (wacht op toets los). Is geen toets meer ingedrukt dan wordt het scannen tot en met de laagste matrixregel voortgezet en verlaten we dit programmadeel in regel 21. We komen dan in een deel terecht waarin opnieuw door scannen het toetsenbord wordt nagelopen, maar nu om na te gaan of een toets wordt ingedrukt (wacht op toets in). Vooraf is in DE het codenummer geladen van de toets die moet worden geselecteerd. In het circuit is de ROM-routine BREAKX opgenomen. Nu KEYINT is uitgeschakeld is het wel nuttig een andere manier te vinden om met de CTRL-STOP-toetsen het programma te kunnen onderbreken. De BREAKX-routine op het adres 00B7 verricht nu deze functie. Zijn tijdens het doorlopen van deze routine de toetsen ingedrukt dan wordt de Carry-FLAG geset, anders gereset. Regel 30 laat een sprong naar het eind van het programma maken. Is een toets ingedrukt dan is de code daarvan in HL geladen. In regel 44 worden HL en DE vergeleken en is het codenummer niet goed dan wordt teruggegaan om te wachten op het indrukken van de juiste toets. We hadden daarbij terug kunnen gaan naar regel 17. Dat is in dit geval niet nodig omdat hoe dan ook alleen het indrukken van de juiste toets het programma laat voortzetten. Aan het eind van

het programma wordt door de regels 49 en 50 in H-KEYI weer de operatiecode voor RET geladen. De interruptroutine KEYINT kan dan weer gewoon zijn werk doen.

9.6. Werken met het cassettedeck.

In veel programma's komt het voor dat een string van karakters van de tape moet worden gelezen of naar de tape moet worden geschreven. We beginnen met dit laatste. Het volgende programma maakt het mogelijk.

```
1                      ;*************
2                      ;* Karakters *
3                      ;*  naar de  *
4                      ;*    tape   *
5                      ;*************
6
7
8                             ORG  0E000H
9                             LOAD 0E000H
10
11
12                     ;Invoeren van de karakters
13
14                     ;Wis het scherm
15  E000 CDC300               CALL 00C3H
16  E003 2100E8               LD   HL,0E800H
17  E006 CD9F00        CHGET: CALL 009FH
18                     ;Is CR ingevoerd?
19  E009 FE0D                 CP   0DH
20  E00B 2818                 JR   Z,CR
21  E00D CDA200        CHPUT: CALL 00A2H
22                     ;Is BS ingevoerd?
23  E010 FE08                 CP   08H
24  E012 2804                 JR   Z,BCKSPC
25                     ;Karakter in het geheugen
26  E014 77                   LD   (HL),A
27  E015 23                   INC  HL
28  E016 18EE                 JR   CHGET
29                     ;BS behandeling
30  E018 3E20          BCKSPC: LD  A,20H
31  E01A CDA200               CALL 00A2H
32  E01D 3E08                 LD   A,08H
33  E01F CDA200               CALL 00A2H
34  E022 2B                   DEC  HL
35  E023 18E1                 JR   CHGET
36                     ;CR en LF op het scherm
37  E025 3E0A          CR:    LD   A,0AH
38  E027 CDA200               CALL 00A2H
39  E02A 3E0D                 LD   A,0DH
40  E02C CDA200               CALL 00A2H
41
42                     ;Karakters naar de tape
43
44                     ;Kanaal naar cassettedeck
45  E02F CDEA00        TAPOON: CALL 00EAH
46
47                     ;Markeer de string
48  E032 0608                 LD   B,8
49  E034 3EEA          MARK1: LD   A,0EAH
50                     ;Teken naar tape
51  E036 C5                   PUSH BC
52  E037 CD54E0               CALL CHNTAP
53  E03A C1                   POP  BC
54  E03B 05                   DEC  B
55  E03C 20F6                 JR   NZ,MARK1
56
57                     ;Stringadres in HL
58  E03E 2100E8               LD   HL,0E800H
59
60                     ;Karakter uit de string
61  E041 7E            ZNDCHR: LD  A,(HL)
62                     ;Laatste karakter?
63  E042 A7                   AND  A
64  E043 2806                 JR   Z,ZNDCR
65                     ;Karakter naar tape
66  E045 CD54E0               CALL CHNTAP
67  E048 23                   INC  HL
68  E049 18F6                 JR   ZNDCHR
69
70                     ;CR naar de tape
71  E04B 3E0D          ZNDCR: LD   A,0DH
72  E04D CD54E0               CALL CHNTAP
73
74                     ;Sluit het kanaal
75  E050 CDF000        TAPOOF: CALL 00F0H
76  E053 C9                   RET
77
78                     ;Subroutine zendt een
79                     ;karakter naar de tape
80
81  E054 E5            CHNTAP: PUSH HL
82  E055 CDED00        TAPOUT: CALL 00EDH
83  E058 E1                   POP  HL
84  E059 C9                   RET
85
86                            END
```

In dit programma is er vanuit gegaan dat eerst een string in het geheugen moet worden geplaatst. De werking van de regels 15 tot en met 40 hebben we dan ook al eens behandeld. Deze regels zijn hier toegevoegd om, zoals bij elk programma in dit boek, het programma zonder meer op zijn werking te kunnen controleren. Het eigenlijke saven vangt aan met regel 45. Om signalen naar de tape te kunnen sturen moet eerst de elektronica die de verbinding met de connector voor het cassettedeck verzorgt, worden ingeschakeld. Het ''kanaal'' naar de tape moet worden *geopend*. Hiervoor wordt de routine TAPOON op het adres 00EA gebruikt (regel 45). Deze routine schakelt tevens de cassettemotor in. Om de string op de tape te markeren worden eerst een aantal gelijke karakters naar de tape gezonden die normaal niet als eerste karakter van een string zullen voorkomen. Hiervoor wordt het karakter voor omega gebruikt, codegetal EAH. Er worden acht van deze karakters voor gebruikt zodat de teller (register b) met 8 wordt geladen (regels 48 en 49). De teller mag niet verloren gaan. Omdat de routine waarmee een karakter naar de tape wordt verzonden, de processorregisters AF BC DE en HL gebruikt moet eerst de teller op de STACK worden bewaard (regels 51 en 53). Het versturen van een karakter naar de tape is hier als subroutine uitgevoerd. Het is de routine op de regels 81 tot en met 84. Het eigenlijke werk doet de

ROM-routine op regel 82. Het is TAPOUT op het adres 00ED. Het ASCII-codeteken van het desbetreffende karakter moet in de Accu zijn geladen. De regels 54 en 55 zorgen ervoor dat acht van de karakters naar de tape worden gezonden. Op zichzelf bevat het programma niets bijzonders meer. Uit de string die we zelf eerst in het geheugen hebben geplaatst wordt, met behulp van de subroutine op regel 81, karakter voor karakter naar de tape gezonden. Als het laatste karakter (de "0") is gevonden wordt, in plaats hiervan, een CR naar de tape gezonden. Als laatste moet het kanaal naar de tape weer gesloten worden. Hiervoor wordt de routine TAPOOF op het adres 00F0 gebruikt (regel 75). Daarmee wordt dan ook de cassettemotor afgeschakeld.

```
1                      ;*************
2                      ;* Karakters *
3                      ;*  van de   *
4                      ;*   tape    *
5                      ;*************
6
7
8                                    ORG  0E000H
9                                    LOAD 0E000H
10
11
12                     ;Kanaal naar de cassette
13 E000 CDE100         TAPION:       CALL 00E1H
14
15                     ;Stringadres in HL
16 E003 2100E8                       LD   HL,0E800H
17
18                     ;Wacht op markering
19 E006 0608           WACHT:        LD   B,08H
20 E008 05             WACHT1:       DEC  B
21 E009 FA19E0                       JP   M,STRCH1
22 E00C C5                           PUSH BC
23 E00D CD3FE0                       CALL ONTCHR
24 E010 C1                           POP  BC
25 E011 FEEA                         CP   0EAH
26 E013 28F3                         JR   Z,WACHT1
27 E015 18EF                         JR   WACHT
28
29                     ;Karakter in stringruimte
30 E017 77             STRCHR:       LD   (HL),A
31 E018 23                           INC  HL
32                     ;Karakter van tape
33 E019 CD3FE0         STRCH1:       CALL ONTCHR
34                     ;Laatste karakter?
35 E01C FE0D                         CP   0DH
36 E01E 20F7                         JR   NZ,STRCHR
37
38                     ;Sluit de string af
39 E020 AF                           XOR  A
40 E021 77                           LD   (HL),A
41
42                     ;Sluit het kanaal
43 E022 CDE700         TAPIOF:       CALL 00E7H
44
45                     ;Wis het scherm
46 E025 CDC300         CLS:          CALL 00C3H
47
48                     ;Print de string
49 E028 2100E8                       LD   HL,0E800H
50 E02B 7E             PRTSTR:       LD   A,(HL)
51 E02C A7                           AND  A
52 E02D 2806                         JR   Z,CR
53 E02F CDA200         CHPUT:        CALL 00A2H
54 E032 23                           INC  HL
55 E033 18F6                         JR   PRTSTR
56 E035 3E0A           CR:           LD   A,0AH
57 E037 CDA200                       CALL 00A2H
58 E03A 3E0D                         LD   A,0DH
59 E03C C3A200                       JP   00A2H
60
61                     ;Subroutine ontvangt een
62                     ;karakter van de tape
63
64 E03F E5             ONTCHR:       PUSH HL
65 E040 CDE400         TAPIN:        CALL 00E4H
66 E043 E1                           POP  HL
67 E044 C9                           RET
68                                   END
```

Het lezen van de karakters vanaf de tape gaat in omgekeerde volgorde. Eerst wordt het kanaal van de tape geopend, maar nu voor het invoeren vanaf de tape. Hiervoor is de routine TAPION op het adres 00E1 (regel 13). Ook nu wordt de cassettemotor ingeschakeld. Het eerste karakter dat van de tape ontvangen moet worden heeft het codegetal EAH. Hierop wordt gewacht in de regels 19 tot en met 27. De subroutine ONTCHR haalt een karakter van de tape. Deze subroutine bevindt zich op de regels 64 tot en met 67 en maakt gebruik van de ROM-routine TAPIN op het adres 00E4. De routine TAPIN gebruikt de registers AF, BC, DE en HL. Blijkt door de vergelijking in regel 25 dat niet het juiste karakter is ontvangen dan wordt teruggegaan naar regel 19 (regel 27). Is wel het juiste karakter ontvangen dan moeten er acht van deze na elkaar volgen. Dit wordt vastgesteld in de lus van de regels 20 tot en met 26. Is het aantal niet correct dan wordt op de volgende serie gewacht. Is het aantal minstens gelijk aan acht dan wordt naar regel 33 gegaan voor het volgende karakter. Het laatste karakter van een string op de tape is een CR en tot dit karakter worden alle ontvangen tekens in de stringruimte geladen. Is CR ontvangen dan wordt de string afgesloten door een "0".
Nu wordt de routine TAPIOF op het adres 00E7 aangeroepen om het kanaal naar de tape weer te sluiten en de cassettemotor uit te schakelen. Uiteindelijk wordt de ontvangen string op de gebruikelijke wijze op het scherm geprint.

9.7. Het invoeren van data in de PSG-registers.

Voor het maken van geluidseffecten en muziek is in de MSX- computer een speciale chip ingebouwd, de Programmable Sound Generator (PSG). Het hierdoor veroorzaakte geluid is hoorbaar via de luidspreker van het gebruikte televisietoestel of via

een versterker die op de audioplug van de computer is aangesloten. Het geluid kan bestaan uit een toon met een bepaalde klankkleur of uit ruis. De PSG kan drie stemmen gelijktijdig en onafhankelijk van elkaar laten klinken. Om met de chip te kunnen werken is enige kennis van begrippen betreffende geluid nodig. Het valt buiten het kader van dit boek om deze begrippen hier te behandelen. Blijkt bij het doorlezen van deze paragraaf dat u nog enige kennis omtrent deze begrippen te kort komt dan kan ik u wijzen op het boek " MSX basic leren programmeren" dat bij uitgeverij De Muiderkring is verschenen. Hierin kunt u precies die basiskennis vinden die u voor het gebruiken van de PSG nodig heeft. Overigens vindt u in deze paragraaf voldoende gegevens om de PSG te kunnen programmeren.
Om de PSG een geluid te kunnen laten veroorzaken heeft deze een aantal gegevens nodig. Per stem moeten de frequentie van het geluid worden ingevoerd en het volume daarvan. Ook moet de vorm van de omhullende worden ingesteld. Hiermee wordt bedoeld hoe het geluid bij het inschakelen "aanzwelt" en later weer wegsterft. De snelheid waarmee dat gebeurt wordt met het "frequentiegetal" van de omhullende aangegeven. De vorm van de omhullende is ook in te voeren. Dit alles kan alleen als de PSG een aantal registers bezit waarin deze gegevens kunnen worden geschreven. In totaal zijn dat veertien registers. Voor de indeling zie lijst 26.

Reg.nr.	Functie
0	LB frequentiegetal stem 1 (LF1)
1	HB frequentiegetal stem 1 (HF1)
2	LB frequentiegetal stem 2 (LF2)
3	HB frequentiegetal stem 2 (HF2)
4	LB frequentiegetal stem 3 (LF3)
5	HB frequentiegetal stem 3 (HB3)
6	Getal voor de ruisfrequentie
7	Voor het kiezen van de stem
8	Volume stem 1
9	Volume stem 2
10	Volume stem 3
11	LB frequentiegetal omhullende
12	HB frequentiegetal omhullende
13	Voor het patroon v.d. omhullende

Lijst 26. De PSG-registers.

Voor het instellen van de frequentie van de toon moet een *"frequentiegetal"* worden ingevoerd. De PSG gaat uit van de periodetijd van de trilling zodat het grootste frequentiegetal de laagste frequentie oplevert. Het grootste getal dat kan worden ingevoerd is 4095 (HB = 15, LB = 255). Per stem zijn daarvoor twee registers nodig. Het laagste getal dat nog enig merkbaar effect sorteert is 11. (HB = 0, LB = 11). Voor de diverse tonen zijn de frequentiegetallen in lijst 27 gegeven.

Voor het instellen van het volume moet in het register voor de desbetreffende stem een getal worden geschreven van 0 tot en met 15. Hoe groter het getal des te sterker het geluid. Met het invoeren van een "0" wordt het desbetreffende kanaal uitgeschakeld (geluidssterkte 0). Moet het geluid reageren op de omhullende (in principe wordt hiermee in een bepaalde tijd de geluidssterkte veranderd) dan moet het register voorzien zijn van het getal 16 (10H).

Om aan te geven welke van de drie stemmen geluid moet produceren moet ook een getal worden ingevoerd. Er is hiervoor een register (register 7) waarvan de twee hoogste bits niet worden gebruikt. Voor elke stem is een bit gereserveerd om de toon in te schakelen (waarde van het bit "0") of uit te schakelen (waarde van het bit "1"). Hiervoor zijn de bits 0, 1 en 2 van het register. De bits 3, 4 en 5 worden gebruikt voor het inschakelen van de ruis. Ook nu moet de waarde van het desbetreffende bit "0" zijn voor het inschakelen en "1" voor het uitschakelen. De indeling van het register is:

7 6	5 4 3	2 1 0	Bitnummer
- -	3 2 1	3 2 1	Kanaalnummer
- -	ruis.	toon.	

Het getal in het register laat zich eenvoudig berekenen door de plaatswaarde van het desbetreffende bit af te trekken van 63. Dat betekent dat als geen enkele stem is ingeschakeld, de inhoud van het register 63 is. Wilt u stem 3 een toon laten veroorzaken (plaatswaarde 4) en stem 2 ruis (plaatswaarde 16) dan moet het getal $63 - 4 - 16 = 43$ naar het register worden geschreven. Met dit register kunt u dus elke stem afzonderlijk in- of uitschakelen.

Ook voor de ruisfrequentie moet een getal worden ingevoerd. Hiervoor is voor de drie stemmen gezamenlijk een register beschikbaar zodat de instelling voor alle drie de stemmen geldt. Het frequentiegetal voor de ruis is weer te vinden in lijst 27.

Noot/ octaaf	n	freq. Hz.	HF	LF	ruis get.
C1	—	33.6	13	101	—
C+1	1	34.6	12	157	—
D1	2	36.7	11	235	—
D+1	3	38.8	11	66	—
E1	4	41.2	10	154	—
F1	5	43.6	10	4	—
F+1	6	46.3	9	113	—
G1	7	49.1	8	232	—
G+1	8	52	8	107	—
A1	9	55	7	242	—
A+1	10	58.3	7	127	—
B1	11	61.6	7	25	—
C2	12	65.2	6	179	—
C+2	13	69.3	6	78	—
D2	14	73.3	5	245	—
D+2	15	77.6	5	161	—
E2	16	82.4	5	77	—
F2	17	87.2	5	2	—
F+2	18	92.6	4	185	—
G2	19	98.1	4	116	—
G+2	20	104	4	52	—
A2	21	110	3	249	—
A+2	22	117	3	192	—
B2	23	123	3	140	—
C3	24	131	3	89	—
C+3	25	139	3	39	—
D3	26	147	2	251	—
D+3	27	155	2	209	—
E3	28	165	2	167	—
F3	29	175	2	129	—
F+3	30	185	2	92	—
G3	31	196	2	58	—
G+3	32	208	2	26	—
A3	33	220	1	253	—
A+3	34	233	1	224	—
B3	35	246	1	198	—
C4	36	261	1	173	—
C+4	37	277	1	148	—
D4	38	293	1	125	—
D+4	39	311	1	104	—
E4	40	330	1	83	—
F4	41	349	1	65	—
F+4	42	370	1	46	—
G4	43	393	1	29	—
G+4	44	416	1	13	—
A4	45	440	0	254	—
A+4	46	466	0	240	—
B4	47	493	0	227	—
C5	48	522	0	214	—
C+5	49	554	0	202	—
D5	50	587	0	191	—
D+5	51	621	0	180	—
E5	52	660	0	170	—
F5	53	698	0	160	—
F+5	54	740	0	151	—
G5	55	785	0	143	—
G+5	56	831	0	135	—
A5	57	880	0	127	—
A+5	58	933	0	120	—
B5	59	985	0	114	—
C6	60	1044	0	107	—
C+6	61	1109	0	101	—
D6	62	1173	0	95	—
D+6	63	1242	0	90	—
E6	64	1319	0	85	—
F6	65	1396	0	80	—
F+6	66	1481	0	76	—
G6	67	1570	0	71	—
G+6	68	1663	0	67	—
A6	69	1760	0	64	—
A+6	70	1865	0	60	—
B6	71	1970	0	57	—
C7	72	2088	0	54	—
C+7	73	2217	0	51	—
D7	74	2347	0	48	—
D+7	75	2484	0	45	—
E7	76	2638	0	42	—
F7	77	2792	0	40	—
F+7	78	3962	0	38	—
G7	79	3140	0	36	—
G+7	80	3326	0	34	—
A7	81	3520	0	32	—
A+7	82	3730	0	30	30
B7	83	3941	0	28	28
C8	84	4175	0	27	27
C+8	85	4434	0	25	25
D8	86	4693	0	24	24
D+8	87	4969	0	23	23
E8	88	5276	0	21	21
F8	89	5583	0	20	20
F+8	90	5923	0	19	19
G8	91	6279	0	18	18
G+8	92	6652	0	17	17
A8	93	7040	0	16	16
A+8	94	7461	0	15	15
B8	95	7882	0	14	14
	96	8351	0	13	13

Lijst 27. Frequentiegetallen.

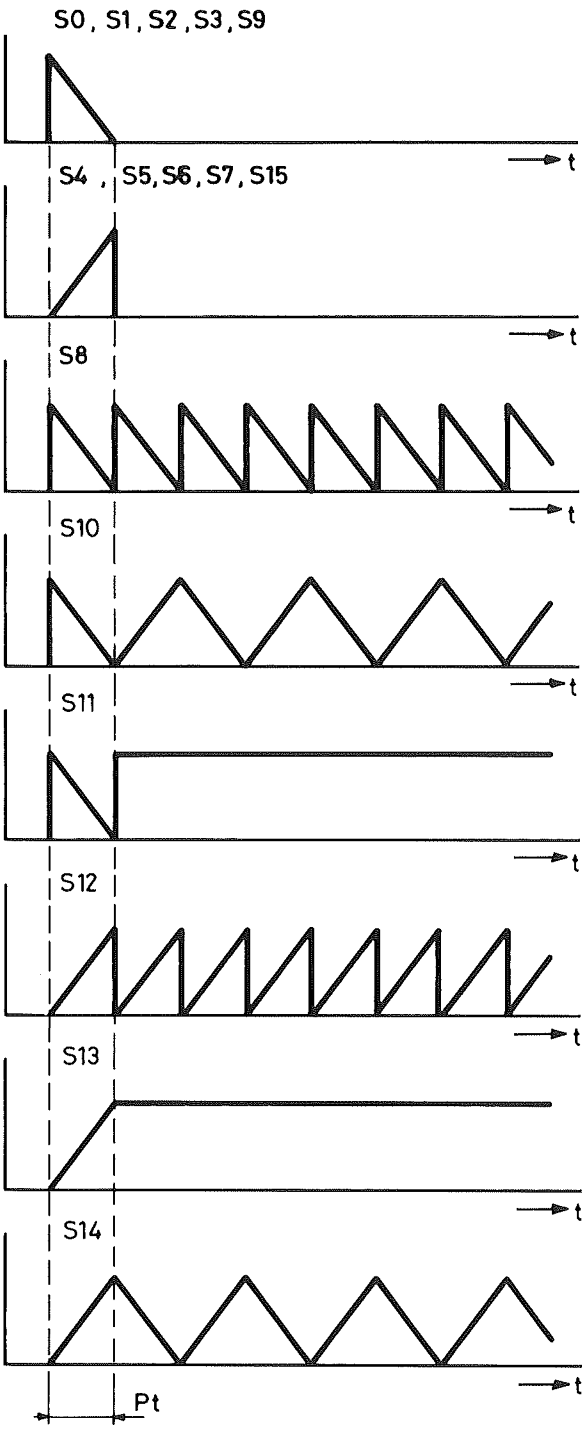


Fig. 69. De diverse omhullingen.

Het instellen van het patroon van de omhullende kan ook gebeuren met het invoeren van een getal. Het is een getal van 0 tot en met 14 en wordt in figuur 69 met S aangegeven. Wordt de gekozen omhullende met S8 aangegeven dan dient u in het desbetreffende register het getal 8 in te voeren.

Het frequentiegetal voor de omhullende kan berekend worden. Hiervoor dient de formule $n = 6991 * P_t$. Hierin is n het frequentiegebied en P_t de tijd die een verandering duurt (aangegeven in figuur 69). Stel dat P_t 0.3 s moet duren dan wordt het frequentiegetal: $n = 6991*0.3 = 2097.3$. Van dit getal moeten de hoge en de lage byte worden bepaald.
HB = INT(2097.3/256) = 8
LB = INT(2097.3-256*8) = 49
Dit zijn de twee getallen die in de desbetreffende PSG-registers moeten worden ingevoerd.

Het invoeren van de gegevens in de PSG-registers gebeurt via bepaalde uitgangspoortregisters. Hiervoor is in de BASIC-ROM een routine beschikbaar zodat niet direct naar de desbetreffende uitgangspoorten moet worden geschreven. Deze routine is in het volgende programma gebruikt. Dit programma veroorzaakt een toon als u op een bepaalde toets drukt. De toon duurt zolang als de toets wordt ingedrukt. Het programma zal in gedeelten worden doorgenomen.

```
                   ;***********
                   ;* Muziek  *
                   ;*    op   *
                   ;* toetsen *
                   ;***********


                               ORG   OE000H
                               LOAD  OE000H


                   ;JP SCRINT op H-KEYI
E000 218CE0        DISINT:     LD    HL,SCRINT
E003 229BFD                    LD    (OFD9BH),HL
E006 3EC3                      LD    A,OC3H
E008 329AFD                    LD    (OFD9AH),A

                   ;Initieer PSG registers

                   ;Kies de stem
                   ;Registernummer in A
E00B 3E07                      LD    A,07H
                   ;Registerinhoud in E
E00D 1E3E                      LD    E,3EH
E00F CD9300        WRTPSG:     CALL  0093H

                   ;LB freq. get. omhullende
E012 3E0B                      LD    A,OBH
E014 1EFF                      LD    E,OFFH
E016 CD9300                    CALL  0093H

                   ;HB freq. get. omhullende
E019 3E0C                      LD    A,OCH
E01B 1E02                      LD    E,02H
E01D CD9300                    CALL  0093H
```

```
36
37                       ;Patroon van de omhullende
38 E020 3E0D                     LD    A,0DH
39 E022 1E0D                     LD    E,0DH
40 E024 CD9300                   CALL  0093H
41
42                       ;Wacht op toets los
43 E027 2609             WCHTTL:  LD    H,09H
44 E029 7C               WACHT1:  LD    A,H
45 E02A 3D                        DEC   A
46                       ;Geen toets in, naar PSGOFF
47 E02B FA38E0                    JP    M,PSGOFF
48 E02E 67                        LD    H,A
49 E02F CD4101                    CALL  0141H
50 E032 FEFF                      CP    0FFH
51 E034 28F3                      JR    Z,WACHT1
52 E036 18EF                      JR    WCHTTL
53
54                       ;Schakel geluid uit
55 E038 3E08             PSGOFF:  LD    A,08H
56 E03A 1E00                      LD    E,00H
57 E03C CD9300                    CALL  0093H
58
```

Aan het begin van het programma wordt op de gebruikelijke wijze de interruptroutine KEYINT uitgeschakeld (regels 13 tot en met 16). Hierna worden de diverse gegevens ingevoerd die voor het klinken van de desbetreffende toon nodig zijn. Ten eerste wordt stem 1 gekozen door in register 7 het getal 62 (3EH) te schrijven. Dit is gevonden met n = 63-1 = 62. De plaatswaarde van het bit voor stem 1 is 1. Het inschrijven gebeurt met de routine WRTPSG op het adres 0093H (regel 25). Het nummer van het register dat moet worden ingeschreven moet in de Accu worden geladen en het getal dat in dat register moet worden geschreven in E. Voor de periodetijd van de omhullende is ongeveer 0.1 s gekozen. De getallen voor de lage byte (255) en de hoge byte (2) worden op overeenkomstige wijze ingevoerd als de stemkeuze. Dat is ook het geval met het patroon van de omhullende. Omhullende S13 (0DH) is gekozen omdat de toon na het inschakelen blijft doorklinken tot hij weer wordt uitgeschakeld. Na het invoeren van deze gegevens wordt gewacht op het loslaten van de eventueel ingedrukte toets. Is geen toets ingedrukt dan wordt het geluid in de regels 55 tot en met 57 uitgeschakeld. Als de toon nog niet was ingeschakeld dan kan uitschakelen geheel geen kwaad. Steeds wordt aan het eind van het programma naar het programmagedeelte "wacht op toets los" teruggegaan. Het geluid dat dan inmiddels met het indrukken van een toets is ingeschakeld, wordt aangehouden tot de desbetreffende toets wordt losgelaten. Dan komen we terecht bij de regels 55 tot en met 57 waarin het getal 00H in het PSG-register 8 wordt geschreven voor geluidssterkte 0.

```
59                       ;Test de CTRL-STOP toetsen
60 E03F CDB700           BREAKX:   CALL  00B7H
61 E042 3842                       JR    C,ENAINT
62                       ;Wacht op toets in
63 E044 2609             SCAN:     LD    H,09H
64 E046 7C               SCAN1:    LD    A,H
65 E047 3D                         DEC   A
66                       ;Geen toets in, opnieuw
67 E048 FA3FE0                     JP    M,BREAKX
68 E04B 67                         LD    H,A
69 E04C CD4101                     CALL  0141H
70 E04F FEFF                       CP    0FFH
71 E051 28F3                       JR    Z,SCAN1
72 E053 6F                         LD    L,A
73
74                       ;Goede toets ingedrukt?
75 E054 11FE01                     LD    DE,01FEH
76 E057 CD2000           DCOMPR:   CALL  0020H
77 E05A 2808                       JR    Z,ZTABAD
78 E05C 11FE00                     LD    DE,00FEH
79 E05F CD2000                     CALL  0020H
80 E062 30DB                       JR    NC,BREAKX
81
```

Na het loslaten van de toets en het uitschakelen van het geluid komen we terecht in het programmagedeelte "wacht op toets in". Eerst is echter gecontroleerd of de CTRL-STOP-toetsen zijn ingedrukt. Dit is bij dit programma zonder meer nodig. Het keert steeds weer terug bij "wacht op toets los" en er zou geen mogelijkheid zijn om het programma te verlaten als de routine BREAKX niet zou worden gebruikt. De bedoeling is dat alleen de toetsen 1 tot en met 8 worden gebruikt. In de regels 75 tot en met 80 is daarom nagegaan of de goede toets wel ingedrukt is. De code voor de ingedrukte toets vinden we in HL. De routine DCOMPR bedient niet alleen de Z-FLAG, maar ook de C-FLAG. Als de inhoud van HL kleiner is dan die van DE wordt de C-FLAG geset, anders gereset. Als u de codegetallen voor de toetsen 1 tot en met 8 uit lijst 25 leest kunt u zien hoe dit programmadeel werkt.

```
82                       ;Zoek het tabeladres
83
84                       ;Beginadres tabel in BC
85 E064 019DE0           ZTABAD:   LD    BC,TABEL
86                       ;Roteer L tot CY=0
87 E067 CB0D             ZTAB1:    RRC   L
88 E069 3004                       JR    NC,LDFREQ
89                       ;Corrigeer het tabeladres
90 E06B 03                         INC   BC
91 E06C 03                         INC   BC
92 E06D 18F8                       JR    ZTAB1
93
94                       ;Freq. getal in PSG reg.
95 E06F 0A               LDFREQ:   LD    A,(BC)
96 E070 5F                         LD    E,A
97 E071 AF                         XOR   A
98 E072 CD9300                     CALL  0093H
99 E075 03                         INC   BC
100 E076 0A                        LD    A,(BC)
101 E077 5F                        LD    E,A
102 E078 3E01                      LD    A,01H
103 E07A CD9300                    CALL  0093H
```

```
104
105                    ;Schakel geluid in
106 E07D 3E08                    LD   A,08H
107 E07F 1E10                    LD   E,10H
108 E081 CD9300                  CALL 0093H
109 E084 18A1                    JR   WCHTTL
```

Met de toetsen 1 tot en met 8 is een toonladder ten gehore te brengen. Hiervoor zijn de tonen C4 tot en met C5 gekozen. De volgende codegetallen (HB-LB) zijn hiervoor nodig:

C4 1-173
D4 1-125
E4 1-83
F4 1-65
G4 1-29
A4 0-254
B4 0-227
C5 0-214

Deze worden na elkaar in geheugenregisters geladen en vormen daarmee een tabel:

Adr. Code
E09D 214
E09E 0
E09F 173
E0A0 1
E0A1 125
E0A2 1
enz.

Per toon zijn twee geheugenplaatsen nodig. De codegetallen voor de hoogste toon (C5) zijn als eerste in de tabel geladen. Daarna volgen in de normale volgorde de andere, C4, D4 E4 enz. De bedoeling is om met behulp van de toetscode het juiste adres in de tabel te vinden. Het beginadres van de tabel wordt in BC geladen. De lage byte van de toetscode bevindt zich in register L. Passen we hierop RRC toe dan schuift het laagstwaardige bit in de Carry. De code in HL (binair) voor de verschillende toetsen is

11111101 1
11111011 2
11110111 3
11101111 4
11011111 5
10111111 6
01111111 7
11111110 8

Uit deze tabel kunt u aflezen hoe vaak RRC L moet worden toegepast om het Carrybit nul te maken. U kunt dan ook vaststellen hoe vaak bij het beginadres 2 is opgeteld door twee keer INC BC (regel 90 tot en met 92). Bij gebruik van toets 8 is na de eerste keer schuiven al de Carry 0, vandaar dat de frequentiegetallen voor C5 als eerste in de tabel moesten worden geplaatst. De frequentiegetallen moeten in register 0 en register 1 van de PSG worden geladen. Dit vindt plaats in de regels 95 tot en met 103. Uiteindelijk wordt het geluid ingeschakeld door het getal 16 (10H) in PSG-register 8 te laden. In regel 109 wordt teruggegaan naar ''wacht op toets los''. De rest van het programma zal u geen problemen geven.

```
110
111                ;Enable KEYINT routine
112 E086 3EC9      ENAINT:    LD   A,0C9H
113 E088 329AFD               LD   (0FD9AH),A
114 E08B C9                   RET
115
116                ;Uitschakelen KEYINT rout.
117
118                ;Corrigeer CALL H-KEYI
119 E08C E1        SCRINT:    POP  HL
120                ;Registers terug uit STACK
121 E08D DDE1                 POP  IX
122 E08F FDE1                 POP  IY
123 E091 F1                   POP  AF
124 E092 C1                   POP  BC
125 E093 D1                   POP  DE
126 E094 E1                   POP  HL
127 E095 08                   EX   AF,AF
128 E096 D9                   EXX
129 E097 F1                   POP  AF
130 E098 C1                   POP  BC
131 E099 D1                   POP  DE
132 E09A E1                   POP  HL
133                ;Return van interrupt
134 E09B ED4D                 RETI
135
136                ;Tabel van frequentieget.
137 E09D D600      TABEL:     DB   214,0
138 E09F AD01                 DB   173,1
139 E0A1 7D01                 DB   125,1
140 E0A3 5301                 DB   83,1
141 E0A5 4101                 DB   65,1
142 E0A7 1D01                 DB   29,1
143 E0A9 FE00                 DB   254,0
144 E0AB E300                 DB   227,0
145
146                           END
```

10. De videoprocessor en -RAM geheugen

10.1. De Video Display Processor.

In paragraaf 7.2 is de functie van de Video Display Processor (VDP) reeds beschreven: dit onderdeel van de computer zorgt er voor dat uw beeldscherm met de nodige karakters wordt beschreven in de door u gewenste kleuren. Verder zorgt de VDP voor het tekenen van de sprites en het bewegen daarvan over het scherm. De VDP werkt geheel onafhankelijk van de Z-80 processor in uw computer en heeft zelf een geheugen ter beschikking. Dit geheugen, de zogenaamde *"VideoRAM"* is in het algemeen 16Kbyte groot. Samen met de VDP wordt op deze manier een computer gevormd die uitsluitend dient voor het beschrijven van het beeldscherm. De data-uitwisseling tussen de VDP en de Z-80 geschiedt via in- en uitgangspoortregisters. Dat is ook het geval met de data-uitwisseling tussen de Z-80 en de VideoRAM. Data-uitwisseling tussen de VDP en deVideoRAM gaat geheel via een eigen bus.

Om karakters op het scherm te kunnen brengen moet de VDP de beschikking hebben over een *schermgeheugen* en een *patroongenerator*. Het karakter op het scherm is opgebouwd uit stippen en het patroon van de stippen is voor elk karakter uit de MSX-karakterset, opgeslagen in de patroongenerator. In het schermgeheugen is voor elke plaats op het scherm een geheugenplaats gereserveerd waarin de ASCCI-codegetallen van de karakters kunnen worden geschreven. De VDP zal om zijn werk te kunnen doen moeten worden voorzien van de nodige gegevens. Om te beginnen zal hij de plaats in de VideoRAM moeten kennen van het schermgeheugen en van de patroongenerator. Ook moet hij weten in welke schermmode hij moet werken.Er zijn vier schermmoden; de veertigkolommode (mode 0), de tweeëndertigkolommode (mode 1), de hoge resolutiemode (mode 2) en de multicolormode (mode 3). Ook moeten de kleur van de karakters (de voorgrondkleur), de kleur van het scherm (de achtergrondkleur) en de kleur van het kader bekend zijn. Daarnaast zijn er nog de gegevens die de VDP nodig heeft bij het tekenen van de sprites. In de VDP zijn voor al deze gegevens een aantal registers (acht) die alleen kunnen worden ingeschreven maar niet kunnen worden uitgelezen. Deze registers worden aangegeven met VDP(0) tot en met VDP(7). Elk register is achtbits. Omdat het uitlezen van deze registers geen effect sorteert is de waarde waarmee ze zijn ingeschreven bewaard in acht registers van de systeemwerkruimte,

RG0SAV	F3DF	[VDP(0)]
RG1SAV	F3E0	[VDP(1)]
RG2SAV	F3E1	[VDP(2)]
RG3SAV	F3E2	[VDP(3)]
RG4SAV	F3E3	[VDP(4)]
RG5SAV	F3E4	[VDP(5)]
RG6SAV	F3E5	[VDP(6)]
RG7SAV	F3E6	[VDP(7)]

Als u dus de inhoud van bijvoorbeeld het VDP(3) register wilt weten dan kunt u daarachter komen door het register RG3SAV op de geheugenplaats F3E2 uit te lezen. Er is nog een negende (werk)register in de VDP. Dit is alleen maar uit te lezen en niet in te schrijven. Het bevat gegevens betreffende het schermbeeld die eventueel belangrijk kunnen zijn voor de programmeur. De VDP registers hebben niet bij elke schermmode precies dezelfde functies. Bij de volgende beschrijving van de registers zal hiermee rekening worden gehouden.

VDP-register 0.

Bit 0. De computer kan gebruikmaken van een externe VDP. In dat geval moet dit bit 1 zijn. Normaal is dit bit 0. Bit 1. Dit bit wordt gebruikt bij de keuze van de schermmode (M2). Bij de hoge resolutiemode (screen 2) moet dit bit 1 zijn. Bij alle andere schermmoden 0. Bit 2 tot en met 7. Deze bits worden niet gebruikt en moeten 0 zijn.

VDP-register 1.

Bit 0. Dit bit heeft alleen invloed bij de schermmoden 1, 2 en 3. Het heeft betrekking op de resolutie van de sprites. Is het bit 0 dan kan elke stip van het spritepatroon tot voor- of achtergrond worden geprogrammeerd. De spritegrootte is dan normaal. Is het bit 1 dan wordt steeds een blokje van 2 × 2 stippen tot voor- of achtergrond geprogrammeerd

(vergrote sprite). Het bit wordt in combinatie met bit 1 gebruikt.
Bit 1. Alleen bij de moden 1, 2 en 3. Dit bit is 1 bij de spritegrootte 16×16 en 0 bij de spritegrootte 8×8. De twee laagste bits van register 1 hebben de volgende functies:

1	0	Bitnummer.
0	0	8×8 sprite, normaal.
0	1	8×8 sprite, vergroot.
1	0	16×16 sprite, normaal.
1	1	16×16 sprite, vergroot.

Bit 2. Wordt niet gebruikt. Moet altijd 0 zijn.
Bit 3. Keuze van de schermmode (M3). Dit bit moet 1 zijn voor schermmode 3, bij alle andere moden moet het 0 zijn.
Bit 4. Keuze van de schermmode (M0). Dit bit moet 1 zijn voor schermmode 0, bij de andere schermmoden 0. Andere bits voor het kiezen van de schermmode zijn er niet. Dat betekent dat voor schermmode 1 bit 1 van register 0 en de bits 3 en 4 van register 1 allemaal 0 moeten zijn.

0	2	3	M
1	0	0	schermmode 0
0	1	0	schermmode 2
0	0	1	schermmode 3
0	0	0	schermmode 1

Bit 5. Disable/enable VDP interrupt. De interrupts zijn uitgeschakeld als dit bit 0 is. Bedenk dat het toetsenbord dan niet meer wordt gescand. Hiermee wordt niet alleen het scannen van het toetsenbord uitgeschakeld, maar ook de werking van de VDP.
Bit 6. Normaal is dit bit 1. Maken we het 0 dan wordt het scherm "blank", dat wil zeggen dat de karakters verdwijnen en de kleur gelijk wordt aan die van het kader. Kan worden gebruikt bij het wisselen van de inhoud van het scherm, dat dan gebeurt met bit 6=0 zodat van de wisseling niets is waar te nemen. Deze "Blanking" is zeer kort waar te nemen bij het wisselen van de schermmode.
Bit 7. Met dit bit wordt de soort van VideoRAM geselecteerd. Bij een 4Kbyte VideoRAM moet het bit 0 zijn. Dat zal hier niet van toepassing zijn. Bij een 16Kbyte VideoRAM is dit bit 1.

VDP-register 2.
In dit register wordt het adres van het schermgeheugen gedefinieerd. Bij een 16Kbyte geheugenblok (16×1024=16384) zijn veertien adreslijnen nodig om een geheugenplaats te adresseren ($2^{14}=16384$). Een binair getal voor een adres moet dan ook veertienbits groot zijn. Dit past niet in een enkel VDP-register. Een geheugenblok voor een schermgeheugen is echter (maximaal) 24×40=960 geheugenplaatsen groot. Er wordt steeds een blok van 1024 plaatsen gereserveerd. De beginadressen kunnen daarom gekozen worden op 0, 1024, 2048, 3072 enz. Het getal dat in het VDP-register 2 wordt geladen is steeds gelijk aan BAD/1024, waarin BAD het beginadres van het schermgeheugen is. Het kleinste getal is 0 en het grootste (16384−1024)/1024=15. Alleen de vier lage bits van het register zullen hiervoor nodig zijn (de lage tetrade). De hoge tetrade wordt niet gebruikt. Normaal is de inhoud bij mode 0 gelijk aan 0, bij mode 1 gelijk aan 6, bij mode 2 ook gelijk aan 6 en bij mode 3 gelijk aan 2.

VDP-register 3.
Zowel bij mode 1 als bij mode 2 is er sprake van een geheugenblok waarin de kleuren van karakters of van stippen op het scherm kunnen worden ingevoerd. Het verplaatsen van het "kleurgeheugen" heeft bij mode 2 geen zin. Het beginadres van het kleurgeheugen bij mode 1 kan in de VideoRAM worden verplaatst met stappen van 64 geheugenplaatsen. Het getal dat in het register moet worden geplaatst is daarom te berekenen met BAD/64. Het kleinste getal is weer 0 en het grootste (16384−64)/64=255. Hiervoor zijn alle bits van het register nodig. Normaal begint het kleurgeheugen op het adres 8192 en bevat het register het getal 128 (80H).In de mode 2 heeft het kleurgeheugen een omvang van 6144 geheugenplaatsen. In deze mode heeft het geen zin om de diverse geheugenblokken te verplaatsen. Het getal dat in het register is geschreven, is gelijk aan 255 en het kleurgeheugen heeft 8192 als beginadres.

VDP-register 4.
In de mode 0 en 1 wordt dit register gebruikt om de plaats van de patroongenerator aan te geven. Het geheugenblok dat voor de patroongenerator nodig is omvat 2048 geheugenplaatsen. Het geheugenblok is met stappen van 2048 adressen te verplaatsen. Het getal dat in het register moet worden geplaatst is BAD/2048. Het kleinste getal is 0 en het grootste (16384−2048)/2048=7. In deze moden worden daarom alleen de drie laagste bits van

het register gebruikt. Bij mode 0 is de inhoud van het register normaal 1 en bij mode 1 normaal 0. In de mode 2 en 3 geeft de inhoud van het register de plaats van het bit map geheugen. Het bit map geheugen is een bijzondere vorm van de patroongenerator. Bij mode 2 is de inhoud van het register 3 en bij mode 3 is de inhoud 0.

VDP-register 5.
Dit register heeft in de mode 0 geen functie. In de andere moden geeft de inhoud het adres van het sprite kenmerkgeheugen. Het geheugenblok hiervoor omvat 128 geheugenplaatsen en kan ook met stappen van 128 adressen worden verplaatst. Het kleinste getal in het geheugen is 0 en het grootste getal is (16384 – 128)/128 = 127 (7FH). Normaal is het getal 54 in het register geplaatst.

VDP-register 6.
Ook dit register heeft in de mode 0 geen functie. In de andere moden geeft de inhoud het adres van het sprite patroongeheugen. Dit geheugen omvat een blok van 2048 geheugenplaatsen en kan ook met stappen van 2048 adressen worden verplaatst. Het kleinste getal in het register is 0 en het grootste (16384 – 2048)/2048 = 7. Normaal bevat het register het getal 7.

VDP-register 7.
In de mode 0 bevat het register de gegevens voor de kleur van het scherm. De lage tetrade van het getal in het register is het nummer van de kleur van de achtergrond (AK) en het kader. De hoge tetrade is het nummer van de kleur van de voorgrond (VK). Het kleurnummer is een getal van 0 tot en met 15. De kleurnummers kunt u vinden in lijst 28. Het getal in het register kan worden berekend met VK × 256 + AK.
In de mode 1, 2 en 3 zijn afzonderlijke kleurgeheugens in gebruik. In dat geval worden alleen de vier lage bits van het register 7 gebruikt voor het noteren van de kleur van het kader.

VDP-register 8.
Dit is het ''read-only'' register van de VDP. Het bevat gegevens over de status van de VDP en over de sprites.

Bit 7. Dit bit is de interrupt FLAG van de VDP. Het wordt geset nadat het volledige raster van het beeldscherm is gescand. Het wordt gereset als dit ''statusregister'' is gelezen of wanneer de VDP extern is gereset.
Bit 6. Dit bit wordt geset als zich vijf of meer dan vijf sprites op dezelfde horizontale lijn bevinden.
Bit 5. Dit is een FLAG die geset wordt als twee sprites elkaar voor meer dan een stip overlappen. Het kan daarom worden gebruikt om botsingen tussen twee sprites vast te stellen.
Bit 4 tot en met bit 0. Hierin wordt een spritenummer opgeslagen als zich meer dan vier sprites op een horizontale lijn bevinden. De VDP laat dan de sprite met de laagste prioriteit (met het hoogste nummer van de vijf die zich op dezelfde lijn bevinden) verdwijnen van het scherm. Het spritenummer hiervan wordt in deze bits opgeslagen (het hoogste nummer is 31).

Code	Kleur	Code	Kleur
0	transparant	8	rood
1	zwart	9	lichtrood
2	middelgroen	10	donkergeel
3	lichtgroen	11	lichtgeel
4	donkerblauw	12	donkergroen
5	lichtblauw	13	magenta
6	donkerrood	14	grijs
7	hemelsblauw	15	wit

Lijst 28. Kleurcodetabel.

10.2. De BASE-registers.
De indeling van de VideoRAM is voor elke mode vastgelegd door de inhoud van de BASE-registers (zie hiervoor het BASIC BASE- statement). Deze hebben bij het inschakelen van de computer hun inhoud gekregen. Er zijn 20 BASE-registers (genummerd van 0 tot en met 19) en er worden per schermmode vijf gebruikt. Zodra door het gebruiken van het SCREEN-statement op een andere schermmode wordt overgegaan worden de VDP-registers 2 tot en met 6 geladen vanuit de bij deze mode behorende BASE-registers. Zolang u in een bepaalde mode aan het werk bent kunt u de BASE-registers voor een andere mode veranderen. U verandert daarmee de indeling van de VideoRAM voor deze mode. Zodra u tot deze mode overgaat gelden de gegevens die op dat moment in de BASE-registers zijn opgeslagen. Het is niet gebruikelijk de indeling van de VideoRAM te veranderen voor de mode waarin gewerkt wordt. Wilt u een indeling veranderen dan dient dat bij BASIC-programma's bij het initiëren aan het begin van het programma te gebeuren, voordat de desbetreffende mode is gekozen. Bij machinetaalprogramma's wordt dat uit-

gevoerd door de VDP-registers in te schrijven terwijl de desbetreffende mode is ingeschakeld. Om geen ordeloos schermbeeld te krijgen is het wenselijk dit tijdens het veranderen ''blank'' te maken. Het is in elk geval noodzakelijk dat in het geheugenblok dat u als patroongenerator aanwijst ook inderdaad de bitpatronen van de karakters te vinden zijn. Er is anders geen enkele mogelijkheid om het beeldscherm te beschrijven. De volgende tabel geeft u de functie van de diverse BASE-registers en hun normale inhoud.

Schermmode 0.
BASE(0) adres schermgeheugen (0)
BASE(1) niet gebruikt
BASE(2) adres patroongenerator (2048)
BASE(3) niet gebruikt
BASE(4) niet gebruikt

Schermmode 1.
BASE(5) adres schermgeheugen (6144)
BASE(6) adres kleurgeheugen (8192)
BASE(7) adres patroongenerator (0)
BASE(8) adres spritekenmerkgeheugen (6912)
BASE(9) adres spritepatroongeheugen (14336)

Schermmode 2.
BASE(10) adres schermgeheugen (6144)
BASE(11) adres kleurgeheugen (8192)
BASE(12) adres bit map geheugen (0)
BASE(13) adres spritekenmerkgeheugen (6912)
BASE(14) adres spritepatroongeheugen (14336)

Schermmode 3.
BASE(15) adres schermgeheugen (2048)
BASE(16) niet gebruikt.
BASE(17) adres bit map geheugen (0)
BASE(18) adres spritekenmerkgeheugen (6912)
BASE(19) adres spritepatroongeheugen (14336)

Door middel van deze gegevens vindt u de indeling van de VideoRAM bij de diverse moden:

Schermmode 0.
00000-00959 schermgeheugen
02048-04095 patroongenerator

Schermmode 1.
00000-02047 patroongenerator
06144-06911 schermgeheugen
06912-07039 spritekenmerkgeheugen
08192-08223 kleurgeheugen
14336-16383 spritepatroongeheugen

Schermmode 2.
00000-06143 bit map geheugen
06144-06911 schermgeheugen
06912-07039 spritekenmerkgeheugen
08192-14335 kleurgeheugen
14336-16383 spritepatroongeheugen

Schermmode 3.
00000-01535 bit map en kleurgeheugen
02948-02815 schermgeheugen
06912-07039 spritekenmerkgeheugen
14336-16383 spritepatroongeheugen.

11. De tekstmode

11.1. De veertigkolommode.

Direct na het inschakelen van de computer staat deze in de veertigkolom tekstmode. Dat houdt in dat maximaal 40 karakters op een schermregel gaan. In de commandomode is dat niet zonder meer mogelijk. Er zijn met het toetsenbord niet meer dan 37 karakters op een regel te krijgen. U weet dat u dat met de functie WIDTH(X) kunt veranderen. Dat is bij het werken met machinetaalprogramma's niet nodig. Omdat u hiermee de geheugenplaatsen in het schermgeheugen direct inschrijft is elke plaats op het scherm (binnen het kader) zonder meer te bereiken. Een BASIC-statement voor het direct inschrijven van de geheugenplaatsen in de VideoRAM is VPOKE. U kunt hiermee uiteraard ook de geheugenplaatsen van het schermgeheugen bereiken. Probeer maar eens:

```
VPOKE 920,2
```

U ziet het bekende maantje verschijnen op de onderste regel van het scherm, op een plaats die u zo zonder meer niet met de cursor kunt bereiken. U weet nu meteen dat 920 (decimaal) een geheugenplaats is die binnen het schermgeheugen valt. Het startadres van het schermgeheugen is dan ook 00H en met 24 regels van 40 kolommen is een geheugenblok nodig van 960 geheugenplaatsen. Stellen we de plaatsen, die ingenomen kunnen worden door de karakters, voor door blokjes dan is de indeling van het scherm zoals in figuur 70.

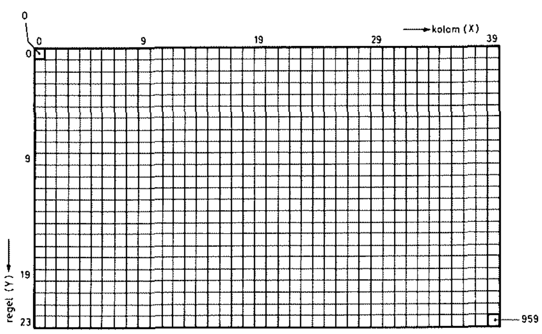


Fig. 70. Beeldschermindeling, veertigkolommode.

Voor elke plaats op het scherm is een geheugenplaats gereserveerd in de VideoRAM. De meest linkse plaats op de bovenste schermregel staat in verbinding met geheugenplaats 0. De tweede plaats op de regel met 1 enz. Dat wil zeggen, als we geen verandering hebben aangebracht desbetreffende het beginadres van het schermgeheugen. Het laatste karakter op de bovenste schermregel heeft betrekking op geheugenplaats 39. De meest linkse plaats op de tweede regel hoort bij geheugenplaats 40. De karakters op de regels corresponderen dus van links naar rechts en van boven naar beneden met een oplopend genummerde geheugenplaats. De schermregels zijn van boven naar beneden van 0 tot en met 23 genummerd en de kolommmen op het scherm van links naar rechts van 0 tot en met 39. Als u een karakter op een bepaalde plaats van het scherm wilt brengen dan kunt u in de bijbehorende geheugenplaats het codeteken van dat karakter schrijven. Wilt u op kolom 14 van regel 7 de letter A geplaatst zien dan schrijft u in de geheugenplaats BAD + 7*40 + 14 het getal 65. BAD is in dit geval 0.
In het algemeen wordt het kolomnummer met X aangegeven en het regelnummer met Y. De formule voor het berekenen van de juiste geheugenplaats wordt dan:

$$P = BAD + Y*40 + X$$

Voor de plaats links onderaan het scherm (kolom 39, regel 23) is X 39 en Y 23. De geheugenplaats die we hiervoor nodig hebben is
$P = 0 + 23*40 + 39 = 959$.

11.2. Bewegende beelden.

Het recept om een bewegend beeld op het scherm te krijgen luidt: print het voorwerp op de gewenste plaats op het scherm, laat het enige tijd staan, wis het en print het opnieuw, echter nu een karakterplaats opgeschoven in de richting die gewenst is. Het volgende programma geeft een voorbeeld van deze techniek. Het laat een bal onder een hoek van 45 graden over het scherm bewegen. Omdat niet direct met de VideoRAM wordt gewerkt maar in

plaats daarvan de subroutine CHPUT voor het printen wordt gebruikt, staat slechts een schermbreedte van 37 kolommen ter beschikking.

```
1                          ;************
2                          ;* Bewegend *
3                          ;* voorwerp *
4                          ;************
5
6
7                          CLS:         EQU   00C3H
8                          ERAFNK:      EQU   00CCH
9                          POSIT:       EQU   00C6H
10                         CHPUT:       EQU   00A2H
11                         ISCNTC:      EQU   00BAH
12                         MOVEX:       EQU   0E800H
13                         MOVEY:       EQU   0E801H
14
15
16                                      ORG   0E000H
17                                      LOAD  0E000H
18
19
20                         ;Wis het scherm
21 E000 AF                              XOR   A
22 E001 CDC300                          CALL  CLS
23                         ;Functietoetsendisplay uit
24 E004 CDCC00                          CALL  ERAFNK
25
26                         ;Laat de bal over het
27                         ;scherm bewegen
28
29                         ;Voor neg. X- en Y-richting
30 E007 3EFE                            LD    A,0FEH
31 E009 3200E8                          LD    (MOVEX),A
32 E00C 3201E8                          LD    (MOVEY),A
33                         ;Begin links bovenaan
34                         ;X=0
35 E00F 2601                            LD    H,1
36                         ;Y=0
37 E011 6C                              LD    L,H
38 E012 1814                            JR    DRWBAL
```

Bij dit programma zijn wat meer de mogelijkheden van het gebruikte assemblerprogramma benut. Bij de aanvang zijn met het pseudocommando EQU een aantal labels gedefinieerd. Dat houdt in dat indien een dergelijke label in het programma wordt aangeroepen, de assembler het hiervoor gedefinieerde adres invult. In regel 7 is het adres van CLS gedefinieerd als 00C3H. In regel 22 vinden we de instructie CALL CLS die door de assembler is uitgewerkt tot de instructiecode CD C3 00. In het programma worden de volgende ROM-routines gebruikt:

CLS. Wist het scherm.

ERAFNK. Schakelt het functietoetsendisplay onderop het scherm uit.

POSIT. Bepaalt de plaats van de cursor op het scherm.

CHPUT. Print een karakter op de plaats die door de cursor wordt aangegeven.

ISCNTC. Gaat na of de CTRL-STOP-toetsen zijn ingedrukt.

Verder worden nog twee geheugenplaatsen gebruikt, MOVEX en MOVEY. Deze zijn om te noteren in welke richting de bal verplaatst moet worden.

Het zal u opgevallen zijn dat twee subroutines kunnen worden gebruikt voor het controleren van de CTRL-STOP-toetsen. De ISCNTC routine wordt gebruikt als het programma zonder meer verlaten kan worden, zonder dat nog bepaalde registers moeten worden gecorrigeerd of een mode moet worden veranderd. Door deze routine komen we direct in de BASIC-commandomode terecht. DE BREAKX routine echter geeft de mogelijkheid, voordat het programma met RET wordt verlaten, nog bepaalde handelingen uit te voeren, zoals het corrigeren van de Stackpointer als aan het begin een PUSH- instructie is uitgevoerd.

Als de bal in de positieve X-richting moet worden verplaatst is in de geheugenplaats MOVEX een 1 geplaatst, anders -1. Hetzelfde geldt voor de Y-richting en MOVEY. Het lijkt vreemd dat in de regels 30 tot en met 32 is gekozen voor een negatieve bewegingsrichting. Omdat de bal echter geheel links bovenaan het scherm begint te bewegen (regels 34 tot en met 38) wordt de richting vanzelf door het programma veranderd in een positieve. Wij kunnen dan vinden dat de eerste plaats op het scherm met $X=0$ en $Y=0$ moet worden aangegeven, de routine POSIT wil hiervoor toch de waarde $H=1$ en $L=1$ ingevoerd hebben. Direct daarna wordt de bal op het scherm geprint.

```
39
40                    ;Verplaats bal over 45 gr.
41
42                    ;Verplaats in X-richting
43 E014 3A00E8        XMOVE:      LD   A,(MOVEX)
44                    ;Bepaal neg. of pos.
45 E017 07                        RLCA
46 E018 3803                      JR   C,MOVENX
47                    ;Positief, verhoog X
48 E01A 24            MOVEPX:     INC  H
49 E01B 1801                      JR   YMOVE
50                    ;Negatief, verlaag X
51 E01D 25            MOVENX:     DEC  H
52
53                    ;Verplaats in Y-richting
54 E01E 3A01E8        YMOVE:      LD   A,(MOVEY)
55                    ;Bepaal neg. of pos.
56 E021 07                        RLCA
57 E022 3803                      JR   C,MOVENY
58                    ;Positief, verhoog Y
59 E024 2C            MOVEPY:     INC  L
60 E025 1801                      JR   DRWBAL
61                    ;Negatief, verlaag Y
62 E027 2D            MOVENY:     DEC  L
```

```
63
64                      ;Teken de bal
65 E028 CDC600          DRWBAL:     CALL POSIT
66 E02B 3EF9                        LD   A,0F9H
67 E02D CDA200                      CALL CHPUT
```

De label XMOVE is de plaats waar het programma steeds weer naar teruggaat om de gehele handeling te herhalen. In dit programmagedeelte wordt de cursor verplaatst in de richting die door MOVEX en MOVEY wordt aangegeven. Eerst wordt de verplaatsing in X-richting bepaald. Hiervoor wordt het getal in MOVEX in A geladen en door RLCA naar links geschoven. Daarbij komt het hoogstwaardige bit in de Carry terecht. Is het getal in MOVEX negatief (– 1) dan is de Carry 1 en wordt door MOVENX de inhoud van H verlaagd. Is het getal in MOVEX positief dan is de Carry 0 en wordt door MOVEPX de inhoud van H verhoogd. Hetzelfde gebeurt in het programmagedeelte YMOVE voor de Y-richting. Omdat zowel de X- als de Y-waarde met 1 is veranderd zal de cursor een plaats in diagonale richting opschuiven. Hierna wordt op deze plaats in de regels 65 tot en met 67 de bal op het scherm geplaatst.

```
68
69                      ;Bepaal of de rand van
70                      ;het scherm is bereikt
71
72                      ;Linker- of rechterrand
73 E030 7C              XLIMIT:     LD   A,H
74 E031 FE01                        CP   01H
75 E033 2804                        JR   Z,XNEG
76 E035 FE25                        CP   25H
77 E037 2008                        JR   NZ,YLIMIT
78                      ;Verander de looprichting
79 E039 3A00E8          XNEG:       LD   A,(MOVEX)
80 E03C ED44                        NEG
81 E03E 3200E8                      LD   (MOVEX),A
82
83                      ;Boven- of onderrand
84 E041 7D              YLIMIT:     LD   A,L
85 E042 FE01                        CP   01H
86 E044 2804                        JR   Z,YNEG
87 E046 FE17                        CP   17H
88 E048 2008                        JR   NZ,VRTRG
89                      ;Verander de looprichting
90 E04A 3A01E8          YNEG:       LD   A,(MOVEY)
91 E04D ED44                        NEG
92 E04F 3201E8                      LD   (MOVEY),A
```

Als de bal aan de rand van het scherm is aangekomen moet de verplaatsing in de X- of de Y-richting worden veranderd. Is de bal aan de linker- of de rechterrand (H = 1 resp. H = 37) dan moet de verplaatsing in X-richting veranderen. Dit gebeurt door het teken van het getal in MOVEX te veranderen. In regel 79 wordt de inhoud van MOVEX in de Accu geladen. Door de instructie NEG wordt de waarde van het getal in de Accu tegengesteld van teken (1 wordt – 1 en – 1 wordt 1). Het nieuwe getal moet weer terug worden geplaatst (regel 81). Voor de Y-richting verloopt de handeling op overeenkomstige wijze.

```
93
94                      ;Vertraag
95 E052 010002          VRTRG:      LD   BC,0200H
96 E055 CDBA00          VRTRG1:     CALL ISCNTC
97 E058 0B                          DEC  BC
98 E059 79                          LD   A,C
99 E05A B0                          OR   B
100 E05B 20F8                       JR   NZ,VRTRG1
101
102                     ;Wis de bal
103 E05D CDC600         SCRBAL:     CALL POSIT
104                     ;Codegetal voor spatie
105 E060 3E20                       LD   A,20H
106 E062 CDA200                     CALL CHPUT
107                     ;Print de bal opnieuw
108 E065 18AD                       JR   XMOVE
109                                 END
```

Omdat de bal op de plaats die hij heeft enige tijd zichtbaar moet zijn wordt nu een vertraging ingelast. Nu hebben we dat al eens eerder nodig gehad, maar er zijn meerdere manieren om een vertraging te veroorzaken. Steeds zal echter eén programmagedeelte voor een aantal keren doorlopen worden. In deze vertraging is in de lus de routine ISCNTC opgenomen zodat de CTRL-STOP toetsen in dit programmagedeelte worden gecontroleerd. Nu is het HL-registerpaar echter niet ter beschikking om door de ROM-routine DCOMPR te worden vergeleken met de inhoud van DE. De oplossing ziet u in de regels 95 tot en met 100. Om tot een vertraging van enige grootte te kunnen komen moet in de lus vaak een routine worden opgenomen die zelf ook enige tijd vraagt en eventueel ook nog een nuttige functie heeft. Hier is dat ISCNTC en in de andere voorbeelden was dat DCOMPR. De lengte van de vertraging is ingevoerd in BC. De instructie DEC BC heeft echter geen invloed op de FLAGS. Dit is opgelost door C in de Accu te laden en een OR-bewerking met B uit te voeren. De Z-FLAG zal slechts geset worden als zowel B als C nul is. Na de vertraging moet de bal, op de plaats waar hij zich nu bevindt, weer worden gewist.
Omdat door het gebruiken van de routine CHPUT voor het tekenen van de bal, de cursor is verschoven, moet hij met CALL POSIT eerst weer op de goede plaats worden gebracht. Daarna wordt op deze plaats het karakter voor spatie geprint. Uiteindelijk gaan we weer terug naar XMOVE om de bal op de volgende plaats te tekenen.

11.3. Het verplaatsen van het schermgeheugen.
Als we uitgebreide handelingen op het scherm verrichten zijn de veranderingen die hiervan het gevolg zijn, altijd op het scherm waarneembaar. Als dat een bezwaar is kunnen we een tweede schermgeheugen gebruiken. Bij mode 0 is de laatste geheugenplaats voor het schermgeheugen 959, terwijl de patroongenerator op geheugenplaats 2048 aanvangt. Hiertussen kan best nog een blok als schermgeheugen worden gereserveerd, het blok van adres 1024 tot en met 1983. Door over te schakelen van het blok op het adres 0 (VDP-register 2 is 0) naar dat op het adres 1024 (VDP-register 2 is 1) wordt de inhoud van het tweede schermgeheugen zichtbaar. Het schermgeheugen dat niet op het scherm zichtbaar is kan dan worden bewerkt. Bewerkingen in een schermgeheugen verlopen altijd via in- en uitgangspoorten en nemen daarom meer tijd in beslag dan bewerkingen in het geheugen dat de processor direct kan adresseren. Daarom is het ook mogelijk de inhoud van het schermgeheugen over te nemen in het "gewone" RAM-geheugen, hierop de bewerkingen uit te voeren en daarna de veranderde inhoud weer terug te brengen in de VideoRAM. In het volgende programma worden twee bewerkingen toegepast: het weergeven van de inhoud van het scherm op de printer (in goed Nederlands: *"Hardcopy"*) en daarop volgend "Scrolling" naar links. Dat betekent dat het gehele beeld bij herhaling een kolom naar links wordt verplaatst. Omdat het vullen van het scherm met de verschoven tekens, hoe snel het ook gaat, mogelijk een minder fraai effect kan geven, wordt de nieuwe inhoud voor het scherm in een schermgeheugen geplaatst dat niet zichtbaar is. Daarna wordt door het veranderen van de inhoud van VDP-register 2 zeer snel naar dit schermgeheugen omgeschakeld.

```
PAGE    1

 1                        ;*************
 2                        ;* Hardcopy  *
 3                        ;*    en     *
 4                        ;* scrolling *
 5                        ;*************
 6
 7
 8                        WRTVDP:     EQU  0047H
 9                        LDIRMV:     EQU  0059H
10                        LDIRVM:     EQU  005CH
11                        LPTOUT:     EQU  00A5H
12                        BREAKX:     EQU  00B7H
13
14
15                                    ORG  0E000H
16                                    LOAD 0E000H
17
18
19                        ;Schermgeheugen naar RAM
20 E000 210000            MVVRSR:     LD   HL,0000H
21 E003 01C003                        LD   BC,03C0H
22 E006 1100E8                        LD   DE,0E800H
23 E009 CD5900                        CALL LDIRMV
24
25                        ;Hardcopy, naar printer
26 E00C 2100E8            HRDCOP:     LD   HL,0E800H
27                        ;Voor 24 regels
28 E00F 0E18                          LD   C,18H
29                        ;Voor veertig karakters
30 E011 0628              HRDCP1:     LD   B,28H
31 E013 7E                HRDCP2:     LD   A,(HL)
32 E014 CDA500                        CALL LPTOUT
33 E017 23                            INC  HL
34 E018 10F9                          DJNZ HRDCP2
35                        ;Einde regel, LF en CR
36 E01A 3E0A              LF:         LD   A,0AH
37 E01C CDA500                        CALL LPTOUT
38 E01F 3E0D              CR:         LD   A,0DH
39 E021 CDA500                        CALL LPTOUT
40 E024 0D                            DEC  C
41 E025 20EA                          JR   NZ,HRDCP1
42
```

Als eerste zijn weer de gebruikte ROM-routines gedefinieerd. Met WRTVDP kan een bepaalde waarde in een VDP-register worden geschreven. Met LDIRMV kan een geheel blok uit het VideoRAM naar de gewone RAM worden overgebracht. LDIRVM doet hetzelfde, maar nu in tegenovergestelde richting. LPTOUT stuurt een karakter naar de printer en BREAKX controleert de CTRL-STOP-toetsen. Eerst zien we het transport van het schermgeheugen naar de RAM. Hiervoor moet het bronadres (het beginadres van het schermgeheugen) in HL worden geladen en het doeladres in DE. De lengte van het blok (960 geheugenplaatsen) gaat in BC. Het overbrengen gaat verder geheel door LDIRMV. Het volgende onderwerp is het maken van de hardcopy. Dit wordt regel voor regel gedaan. Na elke regel moeten een LF en een CR naar de printer zodat de inhoud van het scherm ook in dezelfde vorm door de printer wordt weergegeven. Het beginadres van het geheugenblok is nu E800 en dit wordt in HL geladen. De lengte van een regel (40 karakters) vinden we in B en het aantal regels (24) in C. Het karakter dat naar de printer moet wordt in de Accu geladen, waarna het printen wordt verzorgd door LPTOUT. Voor het volgende karakter wordt HL verhoogd. Dit gaat zo door tot een gehele regel is behandeld. De programmaregels 36 tot en met 39 verzorgen de LF en de CR. Daarna wordt teruggegaan voor de volgende karakterregel en het geheel herhaald tot alle karakterregels zijn behandeld. Heeft u geen printer op de computer aangesloten dan kunt u in elk geval de scrolling laten uitvoeren door de programmaregels 25 tot en

met 41 niet in te voeren in de computer. Voert u ze wel in dan kan het programma alleen maar worden uitgevoerd met een werkende printer.

```
43                  ;Scrolling naar links
44
45                  ;Voor keuze van schermgeh.
46 E027 79                     LD   A,C
47 E028 F5                     PUSH AF
48                  ;HL 1 plaats hoger dan DE
49 E029 2101E8      SCRLL:     LD   HL,0E801H
50 E02C 1100E8                 LD   DE,0E800H
51                  ;Voor 24 regels
52 E02F 0E19                   LD   C,19H
53                  ;Voor 39 karakters
54 E031 0627        SCRLL1:    LD   B,27H
55
56                  ;Bewaar 1e karakter
57 E033 1A                     LD   A,(DE)
58 E034 F5                     PUSH AF
59                  ;Opschuiven v.d. karakters
60 E035 7E          SCRLL2:    LD   A,(HL)
61 E036 12                     LD   (DE),A
62 E037 23                     INC  HL
63 E038 13                     INC  DE
64 E039 10FA                   DJNZ SCRLL2
65                  ;1e kar. in laatste plaats
66 E03B F1                     POP  AF
67 E03C 12                     LD   (DE),A
68
69                  ;Naar de volgende regel
70 E03D 23                     INC  HL
71 E03E 13                     INC  DE
72 E03F 0D                     DEC  C
73 E040 20EF                   JR   NZ,SCRLL1
74
75                  ;Vertraag de scrolling
76
77                  ;Duur v.d. vertr. in BC
78 E042 010020      VRTRG:     LD   BC,2000H
79                  ;Contr. CTRL-STOP toetsen
80 E045 CDB700      VRTR1:     CALL BREAKX
81 E048 3825                   JR   C,BREAK
82 E04A 0B                     DEC  BC
83 E04B 79                     LD   A,C
84 E04C B0                     OR   B
85 E04D 20F6                   JR   NZ,VRTR1
86
```

Het tweede programmadeel is voor de scrolling naar links. Omdat later steeds van schermgeheugen moet worden gewisseld wordt eerst een getal in de Accu geladen en dat in de STACK bewaard. Het is het getal 0 (LD A,C; het vorige programmadeel heeft C 0 gemaakt). Hierna begint de eigenlijke scrolling. Daarvoor geeft HL het bronregister en DE het doelregister aan. Omdat de karakters slechts één plaats per keer opschuiven heeft HL een inhoud die 1 hoger is dan die van DE. Er zijn weer vierentwintig regels en per regel worden 40 karakters behandeld, eerst 39 en daarna nog een. Het meest linkse karakter op een regel zou van het scherm geschoven worden. Dit karakter wordt bewaard (regels 57 en 58). Hierna worden negenendertig karakters naar links geschoven. Dit had ook met de bloktransferinstructie LDIR kunnen gebeuren. Nu is echter het registerpaar BC niet beschikbaar (in C is het aantal karakterregels geladen) zodat DJNZ wat handiger is. Als laatste is het rechtse karakter op de regel een plaats naar links gegaan. Nu wordt de laatste geheugenplaats van de regel voorzien van het karakter dat aan de linkerkant van het scherm geschoven is. Hierdoor zien we aan de rechterkant op het scherm komen wat aan de linkerkant hiervan verdwenen is. Het scherm gaat dus ''rollen''. Als u wilt kunt u in plaats hiervan ook de laatste geheugenplaats van de regel van een nieuw karakter voorzien. Dit geheel herhaalt zich voor alle vierentwintig regels. Als nu geen vertraging zou worden ingelast dan zou alles op het scherm aan u voorbij ''schieten''. Tijdens de vertraging worden de CTRL-STOP-toetsen gecontroleerd, zodat u in elk geval op een nette manier het programma kunt verlaten.

```
87                  ;Data weer naar scherm
88
89                  ;Kies het schermgeheugen
90 E04F F1          SHBAD:     POP  AF
91 E050 F5                     PUSH AF
92 E051 A7                     AND  A
93                  ;Kies tweede schermgeheugen
94 E052 110004                 LD   DE,0400H
95 E055 2803                   JR   Z,MVSRVR
96                  ;Kies eerste schermgeheugen
97 E057 110000                 LD   DE,0000H
98                  ;Data terug in VRAM
99 E05A 2100E8      MVSRVR:    LD   HL,0E800H
100 E05D 01C003                LD   BC,03C0H
101 E060 CD5C00                CALL LDIRVM
102                 ;Schakel over van schermg.
103 E063 F1                    POP  AF
104 E064 EE01                  XOR  01H
105 E066 F5                    PUSH AF
106 E067 47                    LD   B,A
107 E068 0E02                  LD   C,02H
108 E06A CD4700                CALL WRTVDP
109                 ;Terug voor volg. scrolling
110 E06D 18BA                  JR   SCRLL
111
112                 ;Terug naar oude situatie
113 E06F AF         BREAK:     XOR  A
114 E070 47                    LD   B,A
115 E071 0E02                  LD   C,02H
116 E073 CD4700                CALL WRTVDP
117 E076 F1                    POP  AF
118 E077 C9                    RET
119                            END
```

Nu alles op het scherm een plaats naar links is geschoven moet de nieuwe inhoud weer zichtbaar worden gemaakt. Daarvoor moeten we het schermgeheugen in VRAM laden. Omdat tevens van schermgeheugen wordt gewisseld is het nodig het juiste blok te gebruiken. Daarvoor halen we het keuzegetal terug uit de STACK. Is dit 0 dan wordt 0400H het adres van het schermgeheugen in het VRAM, anders 0000H. Dit wordt bepaald met

AND A in regel 92. Op het resultaat van deze bewerking wordt eerst in regel 95 gereageerd. De bewerking LD DE die hiervoor komt heeft geen invloed op de FLAGS.
Dat in elk geval DE de inhoud 0400H heeft gekregen maakt niet uit. Is dit niet de juiste inhoud dan wordt die in regel 97 gecorrigeerd. DE bevat het doeladres. In HL komt het bronadres (E800) en in BC de bloklengte. LDIRVM verzorgt verder het overbrengen van het scherm. Nu moet de VDP worden geschakeld op dit schermgeheugen. Daarvoor moet het keuzegetal weer uit de STACK komen en als dit 0 is moet het VDP-register 2 met 1 worden geladen, terwijl ook het keuzegetal 1 moet worden. Is het keuzegetal 1 dan moet het 0 worden, evenals VDP-register 2. De XOR bewerking met het getal 1 brengt deze wisseling tot stand:

00000000 Keuzegetal
00000001 XOR
____
00000001 Nieuw keuzegetal

00000001 Keuzegetal
00000001 XOR
____
00000000 Nieuw keuzegetal

Door het nieuwe keuzegetal in het VDP-register 2 te laden wordt overgeschakeld op het juiste schermgeheugen. Hiervoor moet het keuzegetal in register B en het VDP-registernummer in het register C worden geschreven. WRTVDP verzorgt het werkelijke inschrijven van het VDP-register.
Zijn de CTRL-STOP-toetsen ingedrukt dan komen we terecht op regel 113. Het doel van dit laatste programmagedeelte is het teruggaan naar de oude situatie. Het schermgeheugen op het VRAM-adres 0000H moet weer gebruikt worden terwijl de STACK gecorrigeerd moet worden met POP AF.
Zorg ervoor dat een en ander op het scherm geschreven is voordat u het programma start. Dan valt er tenminste nog wat te printen en te schuiven!

11.4. De opbouw van de karakters.
Alles wat met een computer te maken heeft is binair en bestaat in principe uit kleine eenheden. Deze eenheden kunnen slechts 0 of 1 zijn. Zo is het ook met de opbouw van het scherm. Het "veld" van het scherm (het gedeelte binnen het kader) is opgebouwd uit 49152 eenheden die allemaal in een bepaalde kleur kunnen oplichten. Omdat deze eenheden elk in een andere kleur kunnen oplichten, kunnen ze worden beschouwd als stippen. Vaak kan worden gesproken van oplichten in een voorgrondkleur of in een achtergrondkleur. Het oplichten van een stip in een voorgrondkleur is steeds het gevolg van een 1 in een geheugenplaats en het oplichten in een achtergrondkleur van een 0 in een geheugenplaats. Door een juiste combinatie van nullen en enen worden de karakters zichtbaar.

Het veld is georganiseerd in 192 stippenregels (dit ter onderscheiding van karakterregels) van 256 stippen per regel. In de veertigkolommode kunnen horizontaal maximaal 40 karakters op een regel en voor elk karakter zijn zes stippen nodig. Dat betekent dat voor de karakters 6*40 = 240 stippen worden gebruikt. Er blijven in de breedte dus 16 stippen ongebruikt (aan beide kanten van het scherm acht).
Verticaal worden per karakter acht stippenregels gebruikt zodat er 192/8 = 24 karakterregels zijn. Hiervan is de onderste regel vaak niet te gebruiken omdat hij voor het functietoetsendisplay wordt gebruikt. Door direct het schermgeheugen in te schrijven kan ook deze regel worden gebruikt.
Uit het voorgaande zou blijken dat voor elk karakter een "rooster" van acht stippen hoog en zes stippen breed wordt gebruikt. Dat is niet waar. Voor elk karakter wordt een rooster van acht stippen hoog en ook *acht* stippen breed toegepast. Dit hangt uiteraard samen met het voor de computer welhaast magische getal 8 en dat slaat weer op het achtbits getal dat in een geheugenplaats past. Omdat toch slechts zes stippen in de breedte ter beschikking zijn worden de karakters "overlappend" naast elkaar geplaatst. Figuur 71 geeft hiervan een voorstelling. Een rooster wordt ook wel een *"matrix"* genoemd en door in deze matrix een aantal stippen in een bepaald patroon te laten oplichten (de voorgrondkleur te laten aannemen) worden de karakters getekend. In figuur 71 is te zien hoe de A en de B worden gevormd. In het algemeen zijn de karakters zo gevormd dat de drie rechtse kolommen "blank" zijn, twee om de overlapping mogelijk te maken en nog een derde om een ruimte te houden tussen de karakters op het scherm. Bij bepaalde karakters is dat niet mogelijk gebleken, zoals bij het maantje dat ontstond door

POKE 920,2

dat nu niet bepaald op een volle maan lijkt. Een ge-

deelte van dit figuur is afgedekt door de spatie naast dit karakter.

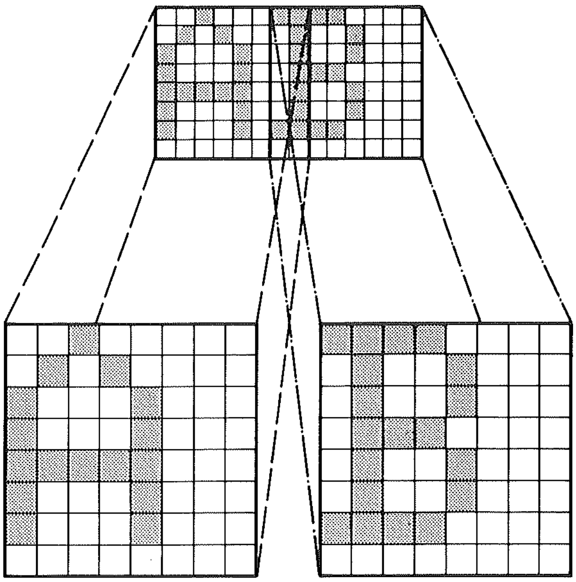


Fig. 71. Het overlappen van de karakters.

De gegevens voor elk karakter worden in geheugenplaatsen opgeslagen. Per karakter zijn 8 geheugenplaatsen nodig. Er zijn 256 verschillende karakters zodat hiervoor een geheugenblok van 8*256 = 2048 geheugenplaatsen gereserveerd moet worden. Dit blok is de "patroongenerator", ook wel de *"karaktergenerator"* genoemd. In de karaktergenerator vinden we steeds blokjes van acht geheugenplaatsen, voor elk karakter een blokje. De eerste geheugenplaats van zo'n blokje bevat de gegevens voor de bovenste regel van het karakter, de tweede geheugenplaats die voor de regel daaronder enz. Uit figuur 71 blijkt dat het blokje voor de A gevuld moet zijn met

00100000
01010000
10001000
11111000
10001000
10001000
00000000

We noemen dit het *"bitpatroon"* voor het karakter A. De onderste regel voor het karakter is "blank" om de karakters op de regels boven elkaar gescheiden te houden.

De gegevens voor de karakters in de karaktergenerator zijn in de volgorde geplaatst die overeenkomt met de codegetallen voor de MSX-karakterset (lijst 23). Het is daarom eenvoudig de gegevens van een bepaald karakter te vinden. De formule hiervoor is:

BAD + 8*ASC

Hierin is BAD het beginadres van de patroongenerator en is ASC de code voor het desbetreffende karakter. U vindt dan het eerste adres van het geheugenblokje voor het karakter. In de veertigkolommode is BAD 2048. Direct na het inschakelen van de computer of direct na het kiezen van de schermmode worden, te beginnen met geheugenplaats 2048, vanuit de BASIC-ROM de gegevens voor de karakters in de patroongenerator geschreven. Wilt u de gegevens voor het karakter C (ASC = 67) dan vindt u het beginadres van het blokje hiervoor met

2048 + 8*67 = 2584.

De opbouw van de karakters is pas goed waar te nemen als ze levensgroot op het beeldscherm geprojecteerd zijn. Dit maakt het volgende programma mogelijk.

```
                    ;************
                    ;*  Opbouw   *
                    ;*   van de   *
                    ;* karakters *
                    ;************


                    WRTVDP:     EQU  0047H
                    RDVRM:      EQU  004AH
                    CHGET:      EQU  009FH
                    CHPUT:      EQU  00A2H
                    ISCNTC:     EQU  00BAH
                    CLS:        EQU  00C3H
                    POSIT:      EQU  00C6H
                    RG4SAV:     EQU  0F3E3H


                                ORG  0E000H
                                LOAD 0E000H


                    ;Lees VDP-register 4
E000 2AE3F3                     LD   HL,(RG4SAV)
                    ;Vermenigvuldig met 2^11
E003 060B                       LD   B,0BH
E005 CB25           MLTPLY:     SLA  L
E007 CB14                       RL   H
E009 10FA                       DJNZ MLTPLY
                    ;Wis het scherm
E00B AF                         XOR  A
E00C CDC300                     CALL CLS
```

```
33                  ;Invoeren van karakter
34 E00F CD9F00      TKCHR:      CALL CHGET
35                  ;Karakter in DE
36 E012 1600                    LD   D,0
37 E014 5F                      LD   E,A
38 E015 CDBA00                  CALL ISCNTC
39                  ;Bewaar HL
40 E018 E5                      PUSH HL
41
42                  ;Bereken adres patr.gen.
43
44                  ;Vermenig. met 2^3 (8*ASC)
45 E019 0603                    LD   B,03H
46 E01B CB23        MTPLY:      SLA  E
47 E01D CB12                    RL   D
48 E01F 10FA                    DJNZ MTPLY
49                  ;Tel op bij BAD
50 E021 19                      ADD  HL,DE
51
```

Voor dit programma zijn weer een aantal routines gebruikt: WRTVDP schrijft een waarde naar een VDP-register, RDVRM leest de inhoud van een geheugenplaats van de VideoRAM, CHGET wacht op het indrukken van een toets en leest het codegetal van het desbetreffende karakter uit de toetsenbordbuffer, CHPUT print een karakter op het beeldscherm, ISCNTC controleert de CTRL-STOP-toetsen, CLS maakt het scherm schoon en POSIT brengt de cursor op de gewenste plaats.

Voor het bepalen van het beginadres van de karaktergenerator lezen we de inhoud van register RG4SAV. De VDP- registers zijn niet rechtstreeks uit te lezen en daarom wordt de inhoud van VDP-register 4, waarvan de waarde een norm is voor het adres van de karaktergenerator, gelezen uit RG4SAV. Om het adres te vinden moet deze waarde met 800H worden vermenigvuldigd. Dit gebeurt door elf keer op L de SLA en op H de RL-instructie toe te passen (regels 23 tot en met 28). De teller is hier B ($0BH = 11_{dec}$). Deze wordt door DJNZ MLTPLY bij elke doorgang met 1 verminderd tot zijn inhoud 0 is. Na het wissen van het scherm wordt het karakter ingevoerd. Het karakter wordt in DE geladen. Op de label TKCHR keert het programma na afloop van het printen van het karakter steeds weer terug. Om het te kunnen verlaten moet daarom de routine ISCNTC worden aangeroepen. Nu moet het adres van het patroon voor het ingevoerde karakter worden berekend. Hiervoor moet eerst het codegetal van het karakter met acht worden vermenigvuldigd. Dit kan met drie keer SLA en RL op resp. E en D. Door AD HL,DE komt de som van de inhoud van DE en HL in HL (BAD + 8*ASC). BAD werd bij het begin van het programma in HL geladen.

```
52                  ;Karakter op het scherm
53
54                  ;Voor 8 karakteregels
55 E022 0E08                    LD   C,08H
56 E024 0600                    LD   B,0
57                  ;Karakt.regel uit VRAM
58 E026 CD4A00      GTCHRG:     CALL RDVRM
59 E029 57                      LD   D,A
60 E02A 23                      INC  HL
61 E02B E5                      PUSH HL
62                  ;Voor de cursorplaats
63 E02C 260F                    LD   H,15
64 E02E 3E04                    LD   A,4
65 E030 80                      ADD  A,B
66 E031 6F                      LD   L,A
67 E032 04                      INC  B
68 E033 CDC600                  CALL POSIT
69
70                  ;Voor acht bits
71 E036 1E08                    LD   E,08H
72 E038 CB22        TKBIT:      SLA  D
73 E03A 3804                    JR   C,TKBIT1
74                  ;Bit=0, spatie
75 E03C 3E20                    LD   A,20H
76 E03E 1802                    JR   TKBIT2
77                  ;Bit=1, wit vierkant
78 E040 3EFF        TKBIT1:     LD   A,0FFH
79 E042 CDA200      TKBIT2:     CALL CHPUT
80                  ;Voor volgend bit
81 E045 1D                      DEC  E
82 E046 20F0                    JR   NZ,TKBIT
83                  ;Voor volgende kar.reg.
84 E048 E1                      POP  HL
85 E049 0D                      DEC  C
86 E04A 20DA                    JR   NZ,GTCHRG
87                  ;Voor volgend karakter
88 E04C E1                      POP  HL
89 E04D 18C0                    JR   TKCHR
90                              END
```

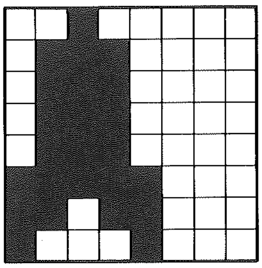
Nu de plaats van het patroon in het karaktergeheugen is berekend wordt dit regel voor regel op het scherm geprint. Voor elke punt van het patroon wordt een karakterplaats op het scherm gebruikt. Voor een voorgrondstip wordt de karakterplaats wit en voor een achtergrondstip wordt de karakterplaats als de achtergrond van het scherm (spatie). De teller voor acht patroonregels is C en B wordt gebruikt voor het bepalen van de plaats van de cursor aan het begin van elke patroonregel op het scherm. In regel 58 wordt een regel van het karakter uit de VideoRAM gelezen met RDVRM. Deze regel wordt in D bewaard. Voor de volgende patroonregel wordt het adres in HL opgehoogd. In de regels 63 tot en met 68 wordt de cursor op zijn plaats gebracht. Gekozen is voor de vijftiende kolom en (voor de bovenste regel van het patroon) de vierde schermregel. Bij de eerste patroonregel vinden we in L het getal 4 (regel 66). Omdat in regel 67 B met 1 wordt verhoogd zal de volgende patroonregel op de vijfde schermregel terechtkomen. Nu wordt het getal in D voor de acht bits onderzocht (teller $E = 8$). Door SLA wordt elk bit een

voor een in de Carry geschoven en afhankelijk van zijn waarde wordt een spatie (bit = 0) of een vierkant (bit = 1) op het scherm geprint. De routine CHPUT zal steeds na het printen van een karakter de cursor een plaats naar rechts laten gaan. Het witte vierkant is een karakter met als code 0FFH. Als een karakterregel is afgehandeld (acht keer schuiven) wordt teruggegaan naar TKBIT voor de volgende patroonregel. Zo worden de acht patroonregels behandeld en staat het karakter in het groot op het scherm. Als het programma is ingevoerd en gestart behoeft u niet anders te doen dan de toets van het gewenste karakter in te drukken.

11.5. De patroongenerator.

Dat de patroongenerator in de VideoRAM is geladen heeft zo zijn voordelen. We kunnen als het nodig is zelf de patronen voor de karakters veranderen. U zou bijvoorbeeld patronen kunnen invoeren voor cursieve karakters. In deze paragraaf krijgt u een voorbeeld van het zelf maken van een karakter.

Voor het ontwerpen van karakters maken we gebruik van een 8 × 8 rooster. Nu hebben we nogal wat karakters ter beschikking in de karaktergenerator, maar een raketje is er nog niet bij. Dat gaan we dan maar eens maken. Zie hiervoor figuur 72.



0 0 1 0 0 0 0 0
0 1 1 1 0 0 0 0
0 1 1 1 0 0 0 0
0 1 1 1 0 0 0 0
0 1 1 1 0 0 0 0
1 1 1 1 1 0 0 0
1 1 0 1 1 0 0 0
1 0 0 0 1 0 0 0

Fig. 72. Bitpatroon van een raket.

In het rooster gaan we de vakjes zwart maken die tezamen het figuur vormen. We houden daarbij rekening met het overlappen van de karakters. Nu gaan we op elke regel voor elk vakje dat niet zwart is een 0 invoeren en voor elk vakje dat zwart is een 1. Er ontstaan op die manier binaire getallen van acht bits die we gaan omzetten tot hexadecimale getallen. Voor het raketje vinden we van boven naar beneden

20, 70, 70, 70, 70, F8, D8, 88

In deze volgorde, dus te beginnen met het getal dat uit de bovenste regel van het rooster is gevormd, brengen we de getallen op de juiste plaats van de patroongenerator. Hiervoor moeten we dit bitpatroon in de plaats brengen van een bitpatroon van de MSX-karakterset. We nemen hiervoor het karakter met het codegetal 128. De plaats in de patroongenerator is te berekenen met BAD + 8*ASC = 2048 + 8*128 = 3072 (0C00H). We gaan daarbij uit van de schermmode 0 waarbij de patroongenerator op het adres 2048 in de VideoRAM aanvangt. Het volgende programma is voor het invoeren van het karakter en het printen daarvan op het scherm.

```
                  ;**************
                  ;* Veranderen *
                  ;*  van een   *
                  ;* bitpatroon *
                  ;**************


                  DCOMPR:     EQU  0020H
                  WRTVDP:     EQU  0047H
                  LDIRVM:     EQU  005CH
                  CHGMOD:     EQU  005FH
                  CHGET:      EQU  009FH
                  CHPUT:      EQU  00A2H
                  BREAKX:     EQU  00B7H
                  CLS:        EQU  00C3H


                              ORG  0E000H
                              LOAD 0E000H


                  ;Invoeren van de kleur
   E000 065E                  LD   B,5EH
   E002 0E07                  LD   C,07H
   E004 CD4700                CALL WRTVDP

                  ;Laadt het bitparoon in
                  ;de patroongenerator

                  ;Beginadres patroon in HL
   E007 212BE0                LD   HL,TABEL
                  ;VRAM adres in DE
   E00A 11000C                LD   DE,0C00H
   E00D 010800                LD   BC,08H
   E010 CD5C00                CALL LDIRVM

                  ;Wis het scherm
   E013 CDC300                CALL CLS
                  ;Invoeren van een karakter
   E016 CD9F00    INCHR:      CALL CHGET
                  ;Controleer op CTRL-STOP
   E019 F5                    PUSH AF
   E01A CDB700                CALL BREAKX
   E01D 3806                  JR   C,RSTSCR
   E01F F1                    POP  AF
                  ;Karakter op beeldscherm
   E020 CDA200                CALL CHPUT
                  ;Voor volgend karakter
   E023 18F1                  JR   INCHR

                  ;Herstel patroongenerator
   E025 F1        RSTSCR:     POP  AF
                  ;Voor schermmode 0
```

```
54 E026 AF                  XOR  A
55 E027 CD5F00              CALL CHGMOD
56 E02A C9                  RET
57
58 E02B 20      TABEL:      DB   20H
59 E02C 70                  DB   70H
60 E02D 70                  DB   70H
61 E02E 70                  DB   70H
62 E02F 70                  DB   70H
63 E030 F8                  DB   0F8H
64 E031 D8                  DB   0D8H
65 E032 88                  DB   088H
66                          END
```

Om te beginnen zien we weer een opsomming van de gebruikte ROM- routines. Ten eerste DCOMPR. Hier hebben we al eens eerder kennis mee gemaakt. Hij dient voor het vergelijken van HL en DE. Met WRTVDP kunnen de registers van de Video Display Processor worden ingeschreven. LDIRVM dient om een blok uit het RAM-geheugen naar de VideoRAM te schrijven. CHGMOD is een routine waarmee de schermmode kan worden gekozen. Met CHGET halen we een karakter uit het toetsenbordbuffergeheugen en met CHPUT laten we dat karakter op het scherm zien. BREAKX controleert de CTRL-STOP-toetsen en CLS maakt het beeldscherm schoon.

Om te beginnen wordt eens wat met de kleur gedaan. Hiertoe moet het kleurnummer in VDP-register 7 worden geladen. Opdat WRTVDP zijn werk kan doen moet het registernummer in C. De kleurnummers worden in B geladen. De achtergrondkleur wordt grijs, kleurnummer 14 (EH). Dit wordt de lage byte voor het getal in B. De voorgrondkleur wordt licht blauw, kleurnummer 5. Hiermee is het getal in B: 5EH. Het laden van het bitpatroon in het patroongeheugen gaat met LDIRM. De lengte van het over te brengen blok bedraagt 8 byte (regel 34). Voor de rest zal dit programmadeel wel geen moeilijkheden geven. Het schermwissen zijn we ook al eens tegengekomen. Daarna volgt het invoeren van een karakter met CHGET. Voor het controleren van de CTRL-STOP toetsen wordt BREAKX gebruikt omdat bij het verlaten van het programma de patroongenerator weer de normale inhoud moet krijgen. BREAKX verandert de inhoud van de Accu, vandaar dat deze met PUSH AF in STACK wordt bewaard. Heeft BREAKX effect dan moet voor het verlaten van het programma eerst nog de STACK-pointer worden gecorrigeerd (regel 52). Bij het verlaten van het programma wordt eerst CHGMOD aangeroepen. Deze routine doet hetzelfde als SCREEN X in BASIC. In dit geval wordt SCREEN 0 uitgevoerd. Dat betekent dat niet alleen de VDP-registers de voor deze mode juiste waarde krijgen, maar ook dat de patroongenerator opnieuw vanuit de BASIC-ROM in de videoRAM wordt geladen. Daarmee is de verandering die door dit programma hierin is aangebracht, teniet gedaan. Zowel de kleur van het scherm als de patroongenerator is weer zoals voorheen. Voor het kiezen van de mode met CHGMOD moet het nummer van de mode in de Accu worden geladen. Hier is deze regel 0 (regel 54).

Het programma kan het beste gestart worden vanuit BASIC met

DEFUSR = &HE000: D = USR(0)

in de schermmode 0. U kunt daarna alle karakters intoetsen en die krijgt u ook allemaal onveranderd op het scherm te zien. Alleen als u de toets 9 met die voor CODE combineert ziet u de raket op het scherm verschijnen.

11.6. De tweeendertigkolommode.

Bij de tweeendertigkolommode is voor elk karakter op het scherm een rooster beschikbaar van 8 bij 8 stippen. De tekens komen daardoor verder uit elkaar te staan terwijl de karakters die de gehele breedte van het rooster nodig hebben ook geheel zichtbaar zijn. Deze mode is daarom erg geschikt om te worden gebruikt met de grafische karakters van de MSX-karakterset en wordt dan ook wel onder de grafische moden gerangschikt. Omdat het behandelen van tekst geheel overeenkomstig is aan die van de veertigkolommode, behandelen we mode 1 in dit hoofdstuk. Er is echter wel een verschil met de behandeling van kleuren. Bij deze mode kennen we een kleurgeheugen. Dit kleurgeheugen laat het (helaas) echter niet toe om elk gewenst karakter in een eigen kleur op het scherm te brengen. Steeds heeft een groep opeenvolgende karakters uit de MSX-karakterset een bepaalde kleur. Het kleurgeheugen vangt aan met het adres 2000H (8192) en de eerste geheugenplaats hiervan bepaalt de kleur van de karakters met de codegetallen 0 tot en met 7. De tweede geheugenplaats bepaalt de kleur van de karakters met het codegetal 8 tot en met 15, enz. Het getal voor de voorgrondkleur van het karakter (van 0 tot en met 15) moet in de hoge tetrade van de geheugenplaats worden geschreven en het getal voor de achtergrondkleur in de lage tetrade. Schrij-

ven we het getal BCH in geheugenplaats 2008H (8192 + 8 = 8200) dan krijgen al de karakters op het scherm met een codegetal tussen 64 (8 × 8) en 71 (inclusief 64 en 71) een lichtgele kleur op een donkergroene achtergrond. Het kleurgeheugen heeft een omvang van 256/8 = 32 bytes.

Het volgende programma demonstreert het gebruiken van twee patroongeneratoren in de schermmode 1. In deze mode bevindt zich de patroongenerator in het blok van 0000H tot en met 07FFH (2047). We gebruiken een tweede patroongenerator die aanvangt op het adres 0800H (2048). Hiervoor moet het getal 1 in VDP-register 4 worden geladen (normaal: 0).

```
1                        ;********************
2                        ;*   Verplaatsen    *
3                        ;*      van de      *
4                        ;* patroongenerator *
5                        ;********************
6
7
8                        DCOMPR:     EQU  0020H
9                        WRTVDP:     EQU  0047H
10                       LDIRMV:     EQU  0059H
11                       LDIRVM:     EQU  005CH
12                       CHGMOD:     EQU  005FH
13                       CHPUT:      EQU  00A2H
14                       BREAKX:     EQU  00B7H
15                       POSIT:      EQU  00C6H
16
17                       VRAM0:      EQU  0000H
18                       VRAM1:      EQU  0800H
19                       BLKLE:      EQU  0800H
20                       SRAM:       EQU  0E800H
21                       RG4SAV:     EQU  0F3E3H
22
23
24                                   ORG  0E000H
25                                   LOAD 0E000H
26
27
28                       ;Schermmode1 inschakelen
29  E000 3E01                        LD   A,01H
30  E002 CD5F00                      CALL CHGMOD
31
32                       ;Patroongenerator in RAM
33
34                       ;Bronadres in HL
35  E005 210000                      LD   HL,VRAM0
36                       ;Doeladres in DE
37  E008 1100E8                      LD   DE,SRAM
38                       ;Bloklengte in BC
39  E00B 010008                      LD   BC,BLKLE
40  E00E CD5900                      CALL LDIRMV
41
42                       ;Inverteer de hoofdletters
43
44                       ;Adres bitpatroon voor A
45  E011 2108EA                      LD   HL,0EA08H
46                       ;Voor 26 karakters
47  E014 06D0                        LD   B,0D0H
48  E016 7E              CMPLMT:     LD   A,(HL)
49  E017 2F                          CPL
50  E018 77                          LD   (HL),A
51  E019 23                          INC  HL
52  E01A 10FA                        DJNZ CMPLMT
53
54                       ;Patroongenerator in VRAM
55  E01C 2100E8                      LD   HL,SRAM
56  E01F 110008                      LD   DE,VRAM1
57  E022 010008                      LD   BC,BLKLE
58  E025 CD5C00                      CALL LDIRVM
59
```

We beginnen met de werking van de gebruikte ROM-routines. DCOMPR voor het vergelijken van HL en DE. WRTVDP voor het schrijven naar een VDP-register. LDIRVM voor het verplaatsen van een geheugenblok van RAM naar VRAM. LDIRMV doet het omgekeerde, van VRAM naar RAM. Met CHGMOD wordt overgeschakeld van schermmode. CHPUT print een karakter op het scherm, BREAKX controleert de CTRL-STOP toetsen en POSIT stuurt de cursor naar de gewenste plaats op het scherm. Dit zijn dus allemaal routines die ook bij de veertigkolommode gebruikt kunnen worden. Behalve het definieren van routines zijn ook nog een aantal geheugenplaatsen benoemd. VRAM0 is het beginadres van de normale patroongenerator in de VideoRAM. VRAM1 is het beginadres van de tweede patroongenerator. Met BLKLE wordt in principe geen geheugenplaats benoemd. Hiermee wordt "bloklengte" bedoeld en deze bedraagt 2048 geheugenplaatsen (0800H). SRAM is een plaats in het "normale" RAM en RG4SAV is de geheugenplaats waarin de inhoud van het VDP-register 4 is opgeslagen.

Het eerste werk van het programma is het schakelen in de mode 1. Er wordt vanuit gegaan dat het programma wordt gestart vanuit de mode 0. Hiermee is tevens het scherm gewist van alle tekst. Bij de nieuwe patroongenerator wordt gebruikgemaakt van de patronen uit de normale patroongenerator. Deze laatste wordt daarvoor eerst overgebracht naar geheugenplaats E800 van het "normale" RAM (SRAM). Hierna worden de patronen voor de hoofdletters "geinverteerd", dat wil zeggen dat de enen in het patroon worden veranderd in nullen en de nullen worden veranderd in enen. De patronen voor de hoofdletters beginnen op het adres 59392 + 8 × 65 = 59912 (EA08H). Dit adres wordt in HL geladen. Er zijn nog steeds 26 letters in ons alfabet. Er moeten daarom 8 × 26 = 208 (D0H) geheugenplaatsen worden behandeld. Eerst wordt de inhoud van een geheugenplaats (een regel van een karakterpatroon) in de Accu geladen waarvan met CPL de bits worden geïnverteerd. Dan wordt het nu verkregen getal weer teruggeplaatst in de geheugenplaats waar het vandaan kwam (regels 48 tot en met 50). Deze werking wordt herhaald voor alle 208 geheugenplaatsen. Na deze bewer-

king wordt de inhoud van het geheugenblok in SRAM overgebracht naar VRAM om daar als tweede patroongenerator te dienen.

```
60                      ;Tekst op het scherm
61
62                      ;Kolomnummer
63  E028 260B                       LD    H,0BH
64                      ;Regelnummer
65  E02A 2E05                       LD    L,05H
66  E02C CDC600                     CALL  POSIT
67                      ;Print de karakters
68  E02F 2170E0                     LD    HL,TABEL
69                      ;Lengte eerste regel
70  E032 0603                       LD    B,03H
71  E034 CD68E0                     CALL  PRINT
72                      ;Tweede regel
73  E037 E5                         PUSH  HL
74  E038 2605                       LD    H,05H
75  E03A 2E06                       LD    L,06H
76  E03C CDC600                     CALL  POSIT
77  E03F E1                         POP   HL
78                      ;Lengte tweede regel
79  E040 060F                       LD    B,0FH
80  E042 CD68E0                     CALL  PRINT
```

Het tweede gedeelte van het programma zorgt ervoor dat er tekst op het scherm wordt geschreven. Daarvoor wordt eerst de cursor met POSIT zijn plaats gewezen. Vanaf deze plaats worden drie karakters op het scherm geprint. De codegetallen van deze karakters zijn gegeven in de tabel aan het eind van het programma. Het gebruikte assemblerprogramma laat het toe hiervoor de karakters zelf tussen de tekens '''' (apostrof) in te voeren. Voor het printen wordt een subroutine gebruikt (PRINT). Deze heeft hiervoor het adres van de tabel nodig (in HL) en de lengte van de te printen string (in B). Op regel 5 van het scherm komen daardoor drie karakters. Op de volgende regel 15 karakters. Ook nu moet eerst de cursor op zijn plaats worden gebracht.

```
81
82                      ;Vertraag
83
84  E045 210000         VRTRG:      LD    HL,0000H
85  E048 11FF7F                     LD    DE,7FFFH
86  E04B 1B             VRTRG1:     DEC   DE
87  E04C CDB700                     CALL  BREAKX
88  E04F 3812                       JR    C,RSTSCR
89  E051 CD2000                     CALL  DCOMPR
90  E054 20F5                       JR    NZ,VRTRG1
91
92                      ;Wissel van patroongen.
93
94                      ;Bepaal welke generator
95                      ;is ingeschakeld
96  E056 3AE3F3                     LD    A,(RG4SAV)
97                      ;Kies andere generator
98  E059 EE01                       XOR   01H
99                      ;Schakel over
100 E05B 0E04                       LD    C,04H
101 E05D 47                         LD    B,A
102 E05E CD4700                     CALL  WRTVDP
103
104                     ;Herhaal
105 E061 18E2                       JR    VRTRG
106
107                     ;Terug naar schermmode 0
108 E063 AF             RSTSCR:     XOR   A
109 E064 CD5F00                     CALL  CHGMOD
110 E067 C9                         RET
111
112                     ;Subr. print de karakters
113 E068 7E             PRINT:      LD    A,(HL)
114 E069 CDA200                     CALL  CHPUT
115 E06C 23                         INC   HL
116 E06D 10F9                       DJNZ  PRINT
117 E06F C9                         RET
118
119 E070 4D5358         TABEL:      DB    'MSX'
120 E073 77657265                   DB    'were'
121 E077 6C647374                   DB    'ldst'
122 E07B 616E6461                   DB    'anda'
123 E07F 6172642E                   DB    'ard.'
124                                 END
```

Nu komt het gedeelte van het programma dat steeds weer wordt herhaald. Eerst volgt een vertraging. Hiervoor worden de inhouden van HL en DE vergeleken door de routine DCOMPR. In de lus van de vertraging is de routine BREAKX opgenomen. Met de CTRL-STOP toetsen kunnen we het programnma dus verlaten. Bij het gebruiken van deze toetsen komen we bij regel 108 terecht. Hier wordt de schermmode 0 ingeschakeld. Dan keert door RET het programma terug naar de BASIC-commandomode. Voordien willen we toch nog wel weten wat de uitwerking van het programma is. Na de vertraging wordt van patroongenerator gewisseld. Hiervoor wordt eerst het getal in RG4SAV opgehaald om te weten welke generator in gebruik is. De inhoud van VDP-register 4 moet 0 of 1 zijn en het veranderen van een 0 in een 1 en een 1 in een 0 komt weer tot stand door XOR 01. De nieuwe waarde wordt naar VDP-register 4 geschreven (regels 100 tot en met 102). Hierna wordt het programmagedeelte vanaf VRTRG herhaald. We krijgen daardoor steeds dezelfde tekst op het scherm te zien, waarbij echter de karakters uit twee verschillende patroongeneratoren worden gebruikt.
De subroutine voor het printen van de karakters op het scherm begint in regel 113. Deze instructievolgorde is al eens eerder aan bod geweest. Het karakter wordt vanuit de TABEL (regel 119) in de Accu geladen. Bedenk wel dat nu twee manieren zijn behandeld voor het printen van een string op het scherm. Bij de eerste manier werd doorgegaan tot het einde van de string werd gevonden. Dit einde was gemarkeerd door een ''0'' (niet het karakter ''0'', ASC = 48, maar het getal 0). De lengte van de string hoeft daarvoor niet bekend te zijn (paragraaf 9.4). Bij de tweede manier wordt de lengte van de string gegeven in het register B.

12. De grafische mode

12.1. Sprites.

Een sprite is een figuur dat zich over het beeldscherm kan verplaatsen zonder dat daardoor de afbeelding op het scherm wordt verstoord. Het is als een ondoorzichtig figuur dat zich voor het scherm langs schijnt te bewegen. Als in het figuur een opening is dan ziet u door de opening de afbeelding (voor- en achtergrond) ter plaatse van het veld. Voor het invoeren van een sprite zijn de volgende gegevens nodig (machinetaal):

a. Het bitpatroon van de sprite. Dit bitpatroon kan het formaat 8×8 maar ook 16×16 hebben.
b. De plaats van de sprite op het scherm, in een X- en een Y- waarde.
c. Het nummer van het bitpatroon.
d. De kleur van de sprite.

Er kan gekozen worden tussen twee formaten van de sprites, een 8×8 formaat zoals bij de karakters en een 16×16 formaat. In het laatste geval beslaat de sprite een oppervlakte van vier karakters. Het is niet mogelijk dat voor elke sprite afzonderlijk te doen. Is de keuze gemaakt voor het 16×16 formaat dan hebben alle sprites op het scherm dit formaat. De bitpatronen moeten in overeenstemming daarmee worden ingevoerd. Wel is het mogelijk de afbeelding van de sprite op het scherm voor zowel de lengte als de breedte twee keer zo groot te laten zijn. Een vergrote 8×8 sprite blijft echter bestaan uit 8×8 stippen. Deze stippen zijn dan meer blokjes van 2×2 stippen. Het "oplossend vermogen is daarmee dus kleiner geworden. Voor de 16×16 sprites geldt hetzelfde. De keuze van de spritegrootte voor wat betreft het aantal stippen en het oplossend vermogen is mogelijk door middel van bit 0 en bit 1 van VDP-register 1 (paragraaf 10.1.).

Voor de plaats van de sprite op het scherm is het orientatiepunt belangrijk. Dit punt ligt een puntenregel boven de sprite en gelijk met het begin van de patroonregels (figuur 73). De X- waarde kan maximaal 255 zijn en de (zinnige) Y-waarde 191. Om de sprite op het scherm te kunnen oriënteren zijn de 255 stippen op een stippenregel met X aangegeven en genummerd van 0 tot en met 255. Van boven naar beneden zijn de stippenregels genummerd van 0 tot en met 191 en met Y aangegeven. P(126,54) is de 127e stip op de 55e stippenregel, gerekend van de meest linkse stip aan de bovenkant van het scherm.

Er kunnen meerdere spritepatronen worden ingevoerd. Elke sprite kan naar wens van één van die patronen gebruikmaken. Het nummer van het gewenste patroon moet bij de gegevens voor de sprite worden ingevoerd. Het patroonnummer kan van 0 tot en met 255 zijn voor sprites van het 8×8 formaat en van 0 tot en met 63 voor het 16×16 formaat.

Voor elke sprite afzonderlijk kan de kleur worden bepaald. Hiervoor moet bij de gegevens voor de sprite een kleurnummer worden ingevoerd. Dit kan een nummer zijn van 0 tot en met 15. De sprite neemt de kleur aan die bij het ingevoerde kleurnummer hoort.

Voor de gegevens van de sprite zijn vier geheugenplaatsen nodig, twee voor de plaats van de sprite op het scherm (de Y- en de X- waarde, in *deze* volgorde) één voor het patroonnummer en één voor het kleurnummer. Deze gegevens vormen het spritekenmerk en zijn in het spritekenmerkgeheugen opgeslagen. Omdat maximaal 32 sprites mogelijk zijn omvat het spritekenmerkgeheugen een blok van 128 geheugenplaatsen. Normaal begint dit blok op het adres 6912 van de VideoRAM. Bij de sprites geldt een bepaalde prioriteit. Daarom zijn ze genummerd, van 0 tot en met 31. De sprite met de hoogste prioriteit heeft het laagste nummer (0). Verder is de prioriteit aflopend met het nummer. De gegevens voor de sprite met het nummer 0 bevinden zich in de vier eerste geheugenplaatsen van het spritekenmerkgeheugen (6912 tot en met 6915). Die voor sprite 1 in de daaropvolgende vier geheugenplaatsen enz. De plaats voor de gegevens van een bepaalde sprite is daarom te berekenen met

$$P = BAD + 4*Sn$$

Hierin is P het adres van het eerste van de vier geheugenplaatsen, BAD het beginadres van het spri-

tekenmerkgeheugen (6912, 1B00H) en Sn het spritenummer. De volgorde van de gegevens is:

Y X Pn c

Hierin zijn X en Y voor de plaats van de sprite op het scherm, is Pn het patroonnummer en c het kleurnummer.

De bitpatronen voor de sprite(s) zijn opgeslagen in het spritepatroongeheugen. Normaal vangt dit aan op de geheugenplaats 14336 (3800H). Voor een sprite van het 8 × 8 formaat zijn acht geheugenplaatsen nodig. Met 256 bitpatronen is dan precies het gehele geheugen gevuld (hoogste adres 16383). Het adres van een bepaald spritepatroon is te vinden met

Sp = BAD + 8*Pn

Hierin is Sp het adres waar het spritepatroon aanvangt, BAD het beginadres van de spritepatroongenerator (14336) en Pn het spritepatroonnummer zoals dat in het kenmerkgeheugen is opgeslagen. Voor een sprite van het 16 × 16 formaat zijn 32 geheugenplaatsen nodig. Ook nu kan de voorgaande formule worden gebruikt voor het bepalen van het beginadres van het spritepatroon. Dat betekent dat in dat geval het spritepatroonnummer dat in het kenmerkregister moet worden geplaatst een veelvoud van 4 is. Voor spritepatroon 0 is het nummer uiteraard ook 0. Voor spritepatroon 1 is het nummer echter 4, voor spritepatroon 2 moet het nummer 8 in het kenmerkregister worden geplaatst, enz.

Het invoeren van het spritepatroon in het spritepatroongeheugen gaat overeenkomstig het invoeren van het bitpatroon van een karakter in de patroongenerator. Bij een 8 × 8 formaat sprite wordt begonnen met het getal dat uit de bovenste regel kan worden gevormd. Dan volgen de getallen van de regels daaronder. Ze worden in opeenvolgende geheugenplaatsen geschreven. Voor een 16 × 16 for-

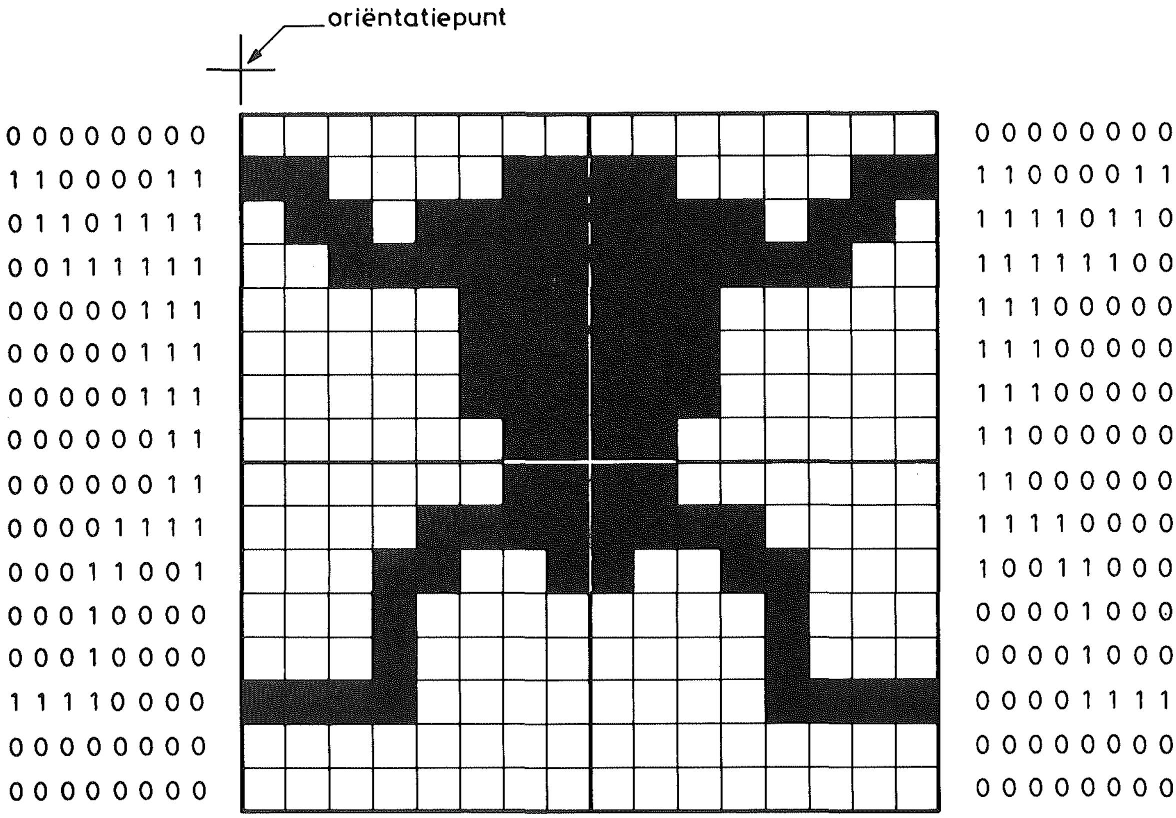


Fig. 73. Spritepatroon voor een kikker.

maat sprite is de gang van zaken overeenkomstig. Er wordt voor het ontwerpen van een dergelijke sprite gebruikgemaakt van een 16 × 16 rooster. De tekening van de sprite wordt hierin aangebracht door het zwart maken van de diverse hokjes. Het patroon in figuur 73 moet een kikker voorstellen. De binaire getallen naast het rooster zijn aan het figuur ontleend. De getallen die links naast het figuur staan moeten het eerst in het geheugen worden geschreven, daarna de getallen die rechts naast het figuur staan. Let op de plaats van het oriëntatiepunt, één puntenregel boven de linkerbovenhoek.

Zodra het bitpatroon in het spritepatroongeheugen is geplaatst, het spritekenmerkgeheugen is ingeschreven en de bits 0 en 1 van het VDP-register 1 de juiste waarde hebben, zal de sprite in de schermmode 1, 2 of 3 op het scherm zichtbaar worden. Het volgende programma laat twee sprites over het scherm bewegen. Sprite 0 is de kikker, sprite 1 is een gekleurd, vierkant vlak.

```
                  ;***********
                  ;* Sprites *
                  ;***********


                  WRTVDP:     EQU  0047H
                  RDVRM:      EQU  004AH
                  WRTVRM:     EQU  004DH
                  FILVRM:     EQU  0056H
                  LDIRVM:     EQU  005CH
                  CHGMOD:     EQU  005FH
                  BREAKX:     EQU  00B7H
                  ERAFNK:     EQU  00CCH
                  RDVDP:      EQU  013EH


                              ORG  0E000H
                              LOAD 0E000H


                  ;Functietoetsdisplay uit
E000 CDCC00                   CALL ERAFNK
                  ;Schermmode 1
E003 3E01                     LD   A,01H
E005 CD5F00                   CALL CHGMOD
                  ;Sprite 16x16, normaal
E008 0E01                     LD   C,1
E00A 06E2                     LD   B,0E2H
E00C CD4700                   CALL WRTVDP

                  ;Spritepatronen in VRAM

                  ;Patroon voor sprite 0
E00F 21A2E0                   LD   HL,TABEL
                  ;Adres spritepatr.geheugen
E012 110038                   LD   DE,3800H
E015 012000                   LD   BC,20H
E018 CD5C00                   CALL LDIRVM
                  ;Patroon voor sprite 1
E01B 3EFF                     LD   A,0FFH
E01D 212038                   LD   HL,3820H
E020 012000                   LD   BC,20H
E023 CD5600                   CALL FILVRM
```

De ROM-routines die in dit programma worden gebruikt zijn: WRTVDP, hiermee kunnen de VDP-registers worden ingeschreven. RDVRM, met deze routine kan een geheugenplaats in de VideoRAM worden gelezen. WRTVRM wordt toegepast als een geheugenplaats in de VideoRAM moet worden ingeschreven. Voor het vullen van een blok geheugenplaatsen in de VideoRAM, met steeds dezelfde waarde, kan FILVRM worden toegepast. LDIRVM brengt een geheugenblok uit het programmageheugen over naar de VideoRAM. Met CHGMOD kan de schermmode worden ingesteld. BREAKX controleert de CTRL-STOP toetsen en ERAFNK laat het functietoetsendisplay van het scherm verdwijnen. Uiteindelijk kunnen we met RDVDP het statusregister van de Video Display Processor (VDP-register 8) lezen.

Het programma vangt aan met het uitschakelen van het functietoetsendisplay en het omschakelen naar schermmode 1. Voor sprites van het 16 × 16 formaat moet VDP-register 1 (C = 1) met het binaire getal 11100010 worden ingeschreven (B = E2H). CALL WRTVDP brengt dat tot stand.

Het spritepatroon voor sprite 0 vinden we in een tabel aan het eind van het programma. Het adres van het bronregister (TABEL) wordt in HL geladen. Het patroon wordt in het eerste geheugenblok van het spritepatroongeheugen gebracht met LDIRVM. Het adres van het doelregister moet daarvoor in DE (3800H). De lengte van het blok is 32 (20H) geheugenplaatsen. Het patroon voor sprite 1 is geheel gevuld. Dat betekent dat alle getallen die ingevoerd moeten worden FFH zijn. Er is een routine waarmee een blok van het VRAM met steeds dezelfde data kan worden gevuld, FILVRM. Deze data moet in de Accu worden geladen. Het doelregister is 3820H, 32 geheugenplaatsen verder (20H) dan 3800H. De bloklengte gaat weer in BC.

```

                  ;Spritekenmerkgegevens

                  ;Adres spritekenm.geheugen
E026 21001B       SPRKMK:     LD   HL,1B00H
                  ;Beginwaarde Y, sprite 0
E029 3E08                     LD   A,08H
E02B CD4D00                   CALL WRTVRM
                  ;Beginwaarde X, sprite 0
E02E 3E00                     LD   A,00H
E030 23                       INC  HL
E031 CD4D00                   CALL WRTVRM
                  ;Patroonnr. sprite 0
E034 3E00                     LD   A,00H
E036 23                       INC  HL
E037 CD4D00                   CALL WRTVRM
                  ;Kleurnr. sprite 0
```

```
61 E03A 3E03                    LD   A,03H
62 E03C 23                      INC  HL
63 E03D CD4D00                  CALL WRTVRM
64                    ;Begingegevens sprite 1
65 E040 3E08                    LD   A,08H
66 E042 23                      INC  HL
67 E043 CD4D00                  CALL WRTVRM
68 E046 3EF0                    LD   A,0F0H
69 E048 23                      INC  HL
70 E049 CD4D00                  CALL WRTVRM
71 E04C 3E04                    LD   A,04H
72 E04E 23                      INC  HL
73 E04F CD4D00                  CALL WRTVRM
74 E052 3E0D                    LD   A,0DH
75 E054 23                      INC  HL
76 E055 CD4D00                  CALL WRTVRM
77
```

De spritekenmerkgegevens die in de VideoRAM moeten worden geladen zijn de beginwaarden voor de plaats van de sprites, de patroonnummmers en de kleurnummers. De beginwaarden voor sprite 0 zijn Y = 8 en X = 0. Het eerste adres in VRAM is 1B00. Hierin komt de Y-waarde. Het inschrijven van een geheugenplaats in de VideoRAM vindt plaats met WRTVRM. De data moet daarvoor in de Accu en het adres in HL. Door INC HL worden de volgende geheugenplaatsen geadresseerd. Achtereenvolgens gaan daarin de X- waarde, het patroonnummer en het kleurnummer (3 voor lichtgroen). De gegevens voor sprite 1 worden op dezelfde manier ingevoerd in het aansluitende blokje van vier geheugenplaatsen. De beginpositie voor sprite 1 is Y = 8 en X = 240 (F0H). Omdat het een 16 × 16 formaat sprite betreft moet het spritepatroongetal voor het patroonnummer 1 gelijk zijn aan 1 × 4 = 4. Voor het kleurnummer is 13 (DH) gekozen.

```
78                    ;Bewegen v.d. sprites
79
80                    ;Vertraag
81 E058 010002        VRTRG:    LD   BC,0200H
82 E05B CDB700        VRTRG1:   CALL BREAKX
83 E05E 383D                    JR   C,RSTSCR
84                    ;Bepaal of de sprites
85                    ;elkaar raken
86 E060 CD3E01                  CALL RDVDP
87 E063 CB6F                    BIT  5,A
88 E065 2807                    JR   Z,VRTRG2
89                    ;Maak sprite 1 onzichtbaar
90 E067 AF                      XOR  A
91 E068 21071B                  LD   HL,1B07H
92 E06B CD4D00                  CALL WRTVRM
93 E06E 0B            VRTRG2:   DEC  BC
94 E06F 79                      LD   A,C
95 E070 B0                      OR   B
96 E071 20E8                    JR   NZ,VRTRG1
97                    ;Verplaats sprite 0
98 E073 21001B                  LD   HL,1B00H
99                    ;Lees Y-waarde sprite 0
100 E076 CD4A00                 CALL RDVRM
101                   ;Verhoog Y-waarde
102 E079 3C                     INC  A
103                   ;Heeft sprite eindstand?
104 E07A FEBF                   CP   0BFH
105 E07C 28AB                   JR   Z,SPRKMK
106                   ;Plaats terug in VRAM
107 E07E CD4D00                 CALL WRTVRM
108                   ;Idem X-waarde
109 E081 23                     INC  HL
110 E082 CD4A00                 CALL RDVRM
111 E085 3C                     INC  A
112 E086 CD4D00                 CALL WRTVRM
113                   ;Verplaats sprite 1
114 E089 23                     INC  HL
115 E08A 23                     INC  HL
116 E08B 23                     INC  HL
117 E08C CD4A00                 CALL RDVRM
118 E08F 3C                     INC  A
119 E090 CD4D00                 CALL WRTVRM
120 E093 23                     INC  HL
121 E094 CD4A00                 CALL RDVRM
122 E097 3D                     DEC  A
123 E098 CD4D00                 CALL WRTVRM
124                   ;Voor de volg. verplaatsing
125 E09B 18BB                   JR   VRTRG
126
```

Het is uiteindelijk de bedoeling dat de sprites over het scherm bewegen. Dit kan als we de X- en Y-waarden van de sprites in het kenmerkgeheugen regelmatig veranderen. Als we ze niet in een flits over het scherm willen zien gaan dan moeten we de sprites op elke plaats even stil laten staan. Er is daarom weer een vertraging ingebouwd. In deze vertraging worden de CTRL-STOP toetsen gecontroleerd. Het is de bedoeling dat een sprite (sprite 1) van het scherm verdwijnt als de figuurtjes elkaar raken. Nu is er in het VDP-statusregister een bit dat geset wordt als twee sprites elkaar raken (paragraaf 10.1). Of de botsing heeft plaats gehad kan worden vastgesteld door het register met CALL RDVDP te lezen en de waarde van bit 5 te bepalen (regels 86 en 87). Nu controleert de VDP de situatie van de sprites elke 20 ms, gedurende een interrupt. De meting in de regels 86 en 87 dient te gebeuren na het optreden van een interrupt. Met een enkele keer meten kan dat wel eens fout gaan. Dit is hier opgelost door de meting op te nemen in de lus van de vertraging. Er wordt dan een groot aantal keren na elkaar gemeten. De sprite wordt onzichtbaar door als kleurnummer 0 in te voeren. Na de vertraging volgt het verplaatsen van de sprites. Hiervoor moeten de X- en de Y-waarden van sprite 0 met een stap worden veranderd. Eerst moet daarvoor de plaats van de sprite worden gelezen uit het kenmerkgeheugen, zowel de Y-waarde als de X-waarde. Dan worden deze veranderd en weer teruggeplaatst. Eerst wordt de Y-waarde van sprite 0 opgehoogd. Als een sprite aan de onderkant van het scherm verdwijnt is het onnodig nog verder door te gaan. Daarom wordt door het vergelijken van de

Y-waarde van sprite 0 met BFH (191) vastgesteld of de sprite al onderaan het scherm is (regel 104). Is dat zo dan wordt gewoon weer opnieuw begonnen met het plaatsen van de sprites aan de bovenkant van het scherm (en het eventueel weer zichtbaar maken van sprite 1). Is de sprite nog niet van het scherm verdwenen dan wordt ook de X-waarde verhoogd. Omdat zowel de X- als de Y-waarde met 1 is opgehoogd gaat de sprite schuin rechts naar beneden over het scherm. Sprite 1 gaat schuin links naar beneden over het scherm. Daarvoor moet de Y-waarde worden opgehoogd en de X-waarde worden verlaagd (regel 122). Het vertragen en het verplaatsen wordt steeds herhaald tot de sprites aan de onderkant van het scherm zijn verdwenen.

```
127                       ;Terug naar schermmode 0
128 E09D AF               RSTSCR:      XOR  A
129 E09E CD5F00                        CALL CHGMOD
130 E0A1 C9                            RET
131
132                       ;Patroontabel sprite 0
133 E0A2 00C3             TABEL:       DB   00H,0C3H
134 E0A4 6F3F                          DB   6FH,3FH
135 E0A6 0707                          DB   07H,07H
136 E0A8 0703                          DB   07H,03H
137 E0AA 030F                          DB   03H,0FH
138 E0AC 1910                          DB   19H,10H
139 E0AE 10F0                          DB   10H,0F0H
140 E0B0 0000                          DB   00H,00H
141 E0B2 00C3                          DB   00H,0C3H
142 E0B4 F6FC                          DB   0F6H,0FCH
143 E0B6 E0E0                          DB   0E0H,0E0H
144 E0B8 E0C0                          DB   0E0H,0C0H
145 E0BA C0F0                          DB   0C0H,0F0H
146 E0BC 9808                          DB   098H,08H
147 E0BE 080F                          DB   08H,0FH
148 E0C0 0000                          DB   00H,00H
149                                    END
```

Het laatste gedeelte van het programma bevat slechts het terugkeren naar schermmode 0 en de patroontabel voor sprite 0.

12.2. De hoge resolutiemode.

Tot nu toe zijn we gewend geweest het scherm te beschrijven met karakters die zijn geplaatst in velden van 8×8 stippen of 8×6 stippen. Zelfs een sprite is in principe als een karakter te beschouwen. Hoge resolutie wil echter zeggen dat elke stip op het scherm apart aan of uit kan worden gezet (de voorgrond- of de achtergrondkleur kan aannemen). In principe hebben we ons daar al mee beziggehouden bij het maken van karakters. In dat geval kunnen we ook bepalen welke stip op het scherm zal oplichten en welke niet, omdat we het karakter zelf kunnen vormen en ook de plaats van het karakter op het scherm zelf kunnen bepalen. Dit principe wordt dan ook toegepast bij de hoge resolutie. Om te beginnen wordt in het schermgeheugen een codegetal geschreven waarmee verwezen wordt naar een karakterpatroon in de patroongenerator. In deze patroongenerator brengen we het bitpatroon van het figuur dat we op het scherm willen zien. Dat betekent dat we nu andersom werken, niet het schermgeheugen wordt ingeschreven met hetgeen we op het beeldscherm willen waarnemen, maar de patroongenerator. De patroongenerator is nu niet met een aantal vaste patronen gevuld en wordt daarom wel het *"bitmapgeheugen"* genoemd.

Het beeldscherm is opgebouwd uit 256×192=49152 stippen. Omdat er acht van in een geheugenplaats passen zijn er 49152/8=6144 geheugenplaatsen nodig om het gehele scherm te kunnen beschrijven. Het bitmapgeheugen moet dan ook deze omvang hebben. De werking blijft echter gelijk aan die van de patroongenerator: een 0 in het schermgeheugen verwijst naar het eerste blokje van acht geheugenplaatsen in het bitmapgeheugen, een 1 naar het tweede blokje van acht geheugenplaatsen enz. Om het laatste blokje van acht geheugenplaatsen in het bitmapgeheugen te kunnen aanwijzen zou het getal (6144−8)/8=767 in het schermgeheugen moeten worden geplaatst. Nu kan in een geheugenplaats geen groter getal dan 255 worden geschreven en er moest dan ook naar een andere oplossing worden gezocht. Deze is gevonden door het bitmapgeheugen in drie delen te splitsen. Het eerste deel correspondeert hierbij met de eerste 256 geheugenplaatsen van het schermgeheugen. Het tweede deel correspondeert met het tweede blok van 256 geheugenplaatsen van het schermgeheugen en het derde deel met het laatste blok. Nu wordt de eerste plaats in het schermgeheugen met een 0 ingeschreven. Dat betekent dat het bitpatroon, dat in het eerste blokje van acht geheugenplaatsen van het bitmapgeheugen is geplaatst, zichtbaar is op de plaats waar, bij de tekstmode, het eerste karakter te zien zou zijn. De tweede plaats in het schermgeheugen wordt ingeschreven met een 1, de derde plaats met een 2 enzovoorts, tot de eerste 256 geheugenplaatsen zijn gevuld. Dat komt neer op acht regels met 32 karakters op het scherm. Het volgende blok in het schermgeheugen wordt op dezelfde wijze behandeld, evenals het derde. Het resultaat is dat het scherm te verdelen is in regels met 32 roosters die elk acht stippen hoog en ook acht stippen breed

zijn (de indeling zoals de karakters bij de 32-kolommode), terwijl elk rooster correspondeert met een blokje van acht geheugenplaatsen van het bitmapgeheugen. Als we niets aan de indeling van de VideoRAM veranderen dan begint in de hoge resolutiemode (schermmode 2) het bitmapgeheugen op het VRAM-adres 0. In fig. 74 is de indeling van het scherm gegeven in roosters. Elk rooster correspondeert met een blokje van acht geheugenplaatsen waarvan de eerste geheugenplaats is voor de bovenste stippenregel van het rooster, de tweede geheugenplaats voor de tweede stippenregel, enz.

Het bepalen van de plaats van een stip op het scherm gaat overeenkomstig het bepalen van de plaats van een sprite. Horizontaal worden de stippen van links naar rechts van 0 tot en met 255 genummerd en met X aangegeven. Van boven naar beneden worden de stippenregels genummerd van 0 tot en met 191 en met Y aangegeven. Daarmee wordt het wel lastig om het adres in het bitmapgeheugen van een bit te vinden dat met een bepaalde stip correspondeert. Over het berekenen van het adres en de plaats van het bit in dat adres behoeven we ons geen zorgen te maken. Er is een ROM-routine die dat voor ons doet.

In deze mode is er ook een uitgebreid kleurgeheugen ter beschikking. Het bevat evenveel geheugenplaatsen als het bitmapgeheugen zodat elke plaats in het kleurgeheugen de kleur bepaalt van acht stippen (naast elkaar) op het scherm. In principe is daarmee de indeling van het kleurgeheugen gelijk aan dat van het bitmapgeheugen. De n^e plaats in het kleurgeheugen correspondeert met dezelfde acht bits als de n^e plaats in het bitmapgeheugen. Het codegetal voor de bits die de voorgrondkleur moeten aannemen wordt in de hoge tetrade van de desbetreffende geheugenplaats geladen. Het codegetal voor de bits die de achtergrondkleur moeten aannemen wordt in de lage tetrade geladen. De begrippen "voorgrond" en "achtergrond" hebben hier niet meer betekenis dan dat ze betrekking hebben op de respectievelijke bits die "1" zijn en de bits die "0" zijn. De stukjes van acht stippen waarin een stippenregel is verdeeld kunnen elk nooit

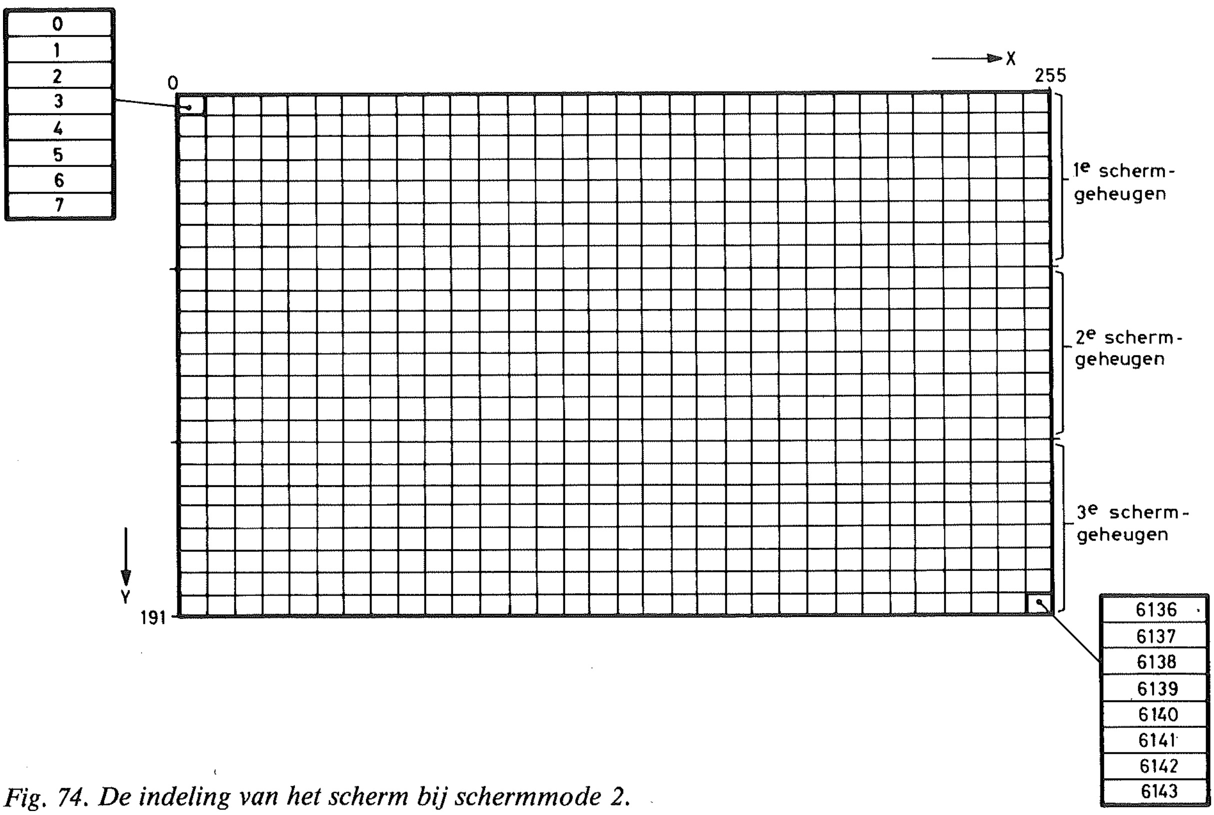


Fig. 74. De indeling van het scherm bij schermmode 2.

meer dan twee kleuren aannemen. Als twee lijnen van verschillende kleuren samen door een dergelijk stukje gaan dan zal een van deze lijnen ter plaatse de kleur van de andere lijn moeten aannemen. Samen met de achtergrondkleur zouden de stippen in drie verschillende kleuren moeten oplichten, en dat gaat nu eenmaal niet.

Het tekenen van horizontale lijnen op het grafische scherm is een eenvoudige zaak. Door uit te gaan van het beginpunt van de lijn kan de X-waarde steeds met 1 verhoogd worden. Het hierna volgende programma zal dat demonstreren. In overeenstemming hiermee is het tekenen van een verticale lijn. Voor een lijn onder vijfenveertig graden moeten de X- en de Y-waarden in gelijke mate worden verhoogd. Moeilijker is het om een willekeurige lijn op het scherm te tekenen. De gebruikelijke formules uit de wiskunde die bij het tekenen zouden kunnen worden toegepast hebben vaak nogal ingewikkelde berekeningen met gebroken getallen tot gevolg . Er is echter een methode waarbij een lijn kan worden getekend met niet meer dan wat eenvoudige optellingen en aftrekkingen met hele getallen.
Als u zich niet interesseert voor de hierna volgende beschouwing, dan slaat u hem eenvoudig over en bepaalt u zich slechts tot de routine die hiervan het gevolg is.

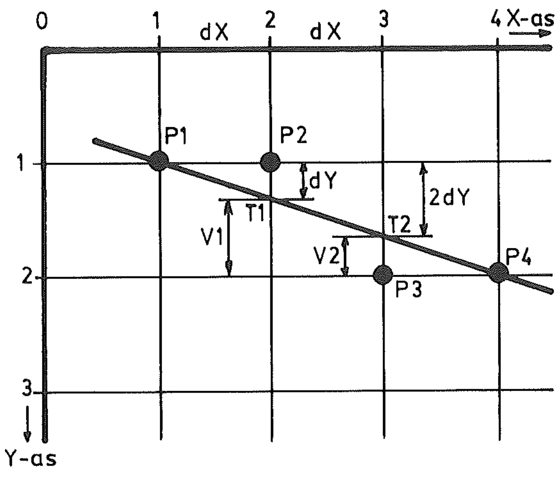


Fig. 75. Het tekenen van een rechte lijn.

Zoals alles op het scherm is ook een lijn opgebouwd uit stippen. Als we voor een lijn alleen die stippen zouden laten oplichten die ook werkelijk samenvallen met de lijn dan zouden wel eens grote afstanden tussen de stippen kunnen ontstaan. In figuur 75 is een lijn getekend waarvan de verandering in X-richting drie keer zo groot is als die in Y-richting. Dat betekent dat alleen de punten P1 en P4 op de lijn vallen. Om nu een zo regelmatig mogelijke lijn te krijgen laten we ook de punten oplichten die het dichtst bij die lijn liggen, de punten P2 en P3. Deze punten vallen samen met de roosterlijnen die over het scherm getekend kunnen worden, uitgaande van een X-as die langs de bovenrand van het veld loopt en een Y-as die langs de linkerkant van het scherm loopt. Op de kruisingen van de roosterlijnen zijn de punten gedacht. Zo zijn er 256 verticale en 192 horizontale roosterlijnen. Denken we ons in dat een cursor zich over het veld verplaatst dan zal deze cursor om van P1 naar P4 te komen de stappen moeten maken zoals getekend in figuur 76. In dit speciale geval zal steeds een stap in X-richting moeten worden gemaakt, maar niet altijd een stap in Y-richting. Of vanuit punt P2 ook een stap in Y-richting moet worden gemaakt hangt af van de lengte van de lijnstukken dY en V1. Nu is V1 groter dan dY en moet geen stap in Y-richting worden gemaakt. Een roosterlijn verder is V2 kleiner dan 2dY en moet wel een stap in Y-richting worden gemaakt. Het gaat dus om de verhouding tussen dY en V. Als we er vanuit gaan dat een stap in elke richting met 1 aangegeven kan worden dan is dY 1/3 en V1 2/3. Dat is lastig want dat zijn gebroken getallen. Omdat verhoudingen niet veranderen als we ze met een bepaald getal vermenigvuldigen kunnen we met een veelvoud van 1/3 en 2/3 werken.

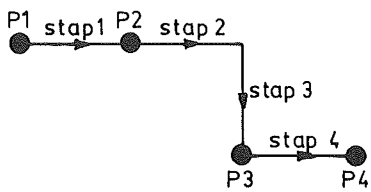


Fig. 76. Het verplaatsen van de cursor.

Het beginpunt van de lijn wordt aangegeven met X1,Y1 en het eindpunt met X2,Y2. Het verschil tussen X2 en X1 wordt gevormd door n-stukjes

dX: n*X = X2 − X1. Het verschil tussen Y2 en Y1 zijn n-stukjes dY: n*dY = Y2 − Y1. Merk op dat dY kleiner is dan dX. Zeker is dat n*dY een heel getal is.
De beslissing of een stap in Y-richting gemaakt moet worden laten we afhangen van het verschil tussen V1 en dY voor de eerste stap, van V2 en 2dY voor de tweede stap enz. Vanuit P1: dY − V1 is negatief zodat geen stap in Y-richting moet worden gemaakt. Vanuit P2: 2dY − V2 is positief zodat nu *ook* een stap in Y- richting moet worden gemaakt.

1[e] stap:
SY1 = dY − V1. Bij een regelmatig rooster is de stap in X-richting even groot als de stap in Y-richting: SY1 = dY − (dX − dY). SY1 = 2dY-dX

2[e] stap:
SY2 = 2dY − V2 = 2dY − (dX − 2dY).
SY2 = 4dY-dX
SY2 = SY1 + 2dY

Hieruit volgt dat voor elke stap in X-richting SY kan worden berekend door de voorgaande waarde van SY te vermeerderen met 2dY.
Is een stap in Y-richting gemaakt dan verandert de situatie (P3). We kunnen dan niet uitgaan van de roosterlijn door P3 maar moeten nu uitgaan van het snijpunt T2.

4[e] stap:
dY3 = dY − V3 = dY − (dX − 2dY) = 3dY − dX
V3 = dX − dY3 = 2dX − 3dY
SY4 = dY3 − V3 = 6dY − 3dX
SY4 = SY2 + 2dY − 2dX

De derde stap was een sprong in Y-richting waarvoor dus kennelijk geldt:

SY3 = SY2 − 2dX

en:

SY4 = SY3 + 2dY

Het resultaat is als volgt samen te vatten:
na een stap in X-richting: SY = SY + 2dY,
na een stap in Y-richting: SY = SY − 2dX.

Omdat de conclusie voor een sprong alleen maar afhangt van het teken van SY (positief of negatief) en niet van de werkelijke grootte, kan dX berekend worden met (X2 − X1) en dY met (Y2 − Y1).
Het programma dat geschreven moet worden moet de "cursor" een aantal stappen laten maken (in X-richting *plus* in Y-richting) dat gelijk is aan dX + dY, omdat dX het aantal stappen in X-richting aangeeft en dY het aantal stappen in Y-richting. Na elke stap in X-richting zal een stip op het scherm "aan" moeten worden gezet.

De voorgaande beschouwing is voor een lijn waarvan dY kleiner is dan dX. Het omgekeerde kan natuurlijk ook. In dat geval moet altijd een stap in Y-richting worden gemaakt terwijl de stap in X-richting afhangt van het teken van SX, die op een overeenkomstige manier als SY kan worden berekend door X en Y in de voorgaande berekening te verwisselen. Voor gevallen waarbij de waarden van dX en/of dY negatief uitvallen (bij de berekeningen moeten steeds de *absolute* waarden van dX en dY worden genomen) moeten we rekening houden met de richting waarin we de cursor verplaatsen.

Het volgende programma demonstreert een aantal mogelijkheden van schermmode 2.

```
                  ;*************
                  ;* Grafische *
                  ;* functies  *
                  ;*************


                  DISSCR:     EQU  0041H
                  ENASCR:     EQU  0044H
                  FILVRM:     EQU  0056H
                  CHGMOD:     EQU  005FH
                  GRPPRT:     EQU  008DH
                  BREAKX:     EQU  00B7H
                  RIGHTC:     EQU  00FCH
                  DOWNC:      EQU  010BH
                  MAPXYC:     EQU  0111H
                  SETATR:     EQU  011AH
                  SETC:       EQU  0120H


                              ORG  0E000H
                              LOAD 0E000H


                  ;Naar schermmode 2
E000 3E02                     LD   A,02H
E002 CD5F00                   CALL CHGMOD
                  ;Maak het scherm blank
E005 CD4100                   CALL DISSCR
                  ;Licht-gele achtergrond
E008 3E0B                     LD   A,0BH
E00A 210020                   LD   HL,2000H
E00D 010018                   LD   BC,1800H
E010 CD5600                   CALL FILVRM
                  ;Schakel het scherm in
E013 CD4400                   CALL ENASCR
```

Zoals gebruikelijk volgt eerst een opsomming van de werking van de gebruikte ROM-routines.
DISCCR maakt het scherm "blank en ENASCR zorgt daarna ervoor dat er weer het een en ander zichtbaar wordt. FILVRM vult een geheugenblok van de VideoRAM met eenzelfde, bepaalde waarde. CHGMOD verzorgt het omschakelen naar een gegeven schermmode. GRPPRT is een routine die het mogelijk maakt om karakters op het scherm te schrijven in de hoge resolutiemode. De CTRL-STOP-toetsen controleert BREAKX. Er is een cursor denkbaar die de plaats van een stip op het hoge resolutiescherm aanwijst. Deze stip kan dan met SETC worden "aangezet" De routine RIGHTC maakt het mogelijk deze cursor een plaats naar rechts te doen gaan. Met DOWNC gaat de denkbeeldige cursor een plaats naar beneden. MAPXYC wordt gebruikt om deze cursor op een bepaalde plaats van het scherm te brengen. Voor lijnen die met ROM-routines op het scherm worden getekend moet een kleurnummer worden ingevoerd. Dit kan met de routine SETATR.
Als eerste wordt in het programma naar schermmode 2 omgeschakeld. Daarna wordt de kleur van de achtergrond in het kleurgeheugen geladen. Dit gebeurt door het kleurnummer 0BH (11) voor een lichtgele (B) achtergrond en een doorzichtige voorgrond (0) in te voeren. Daarom worden alle geheugenplaatsen van het schermgeheugen met dit kleurnummer ingeschreven. Hiervoor gebruiken we de routine FILVRM waarvoor het beginadres van het geheugenblok in HL moet worden geladen en de lengte van het blok (6144 geheugenplaatsen, 1800H) in BC. Het kleurnummer gaat in A en het aanroepen van FILVRM is daarna genoeg om het gehele geheugenblok met dit kleurnummer te vullen. FILVRM werkt (uiteraard) alleen in de Video-RAM. Dit vullen gaat snel maar is desondanks op het scherm waarneembaar. Daarom wordt eerst het scherm blank gemaakt. Na het vullen van het geheugenblok met FILVRM wordt de inhoud van het scherm weer zichtbaar met ENASCR.

```
37                   ;Plaats CRSR op X=47, Y=17
38 E016 212F00                  LD   HL,002FH
39 E019 22B7FC                  LD   (0FCB7H),HL
40 E01C 211100                  LD   HL,0011H
41 E01F 22B9FC                  LD   (0FCB9H),HL
42
43                   ;Kleur v.d. tekst
44 E022 3E06                    LD   A,06H
45 E024 32E9F3                  LD   (0F3E9H),A
46
47                   ;Tekst op het scherm
48 E027 21DAE0                  LD   HL,TABEL
49 E02A 0613                    LD   B,13H
50 E02C 7E          CHOSCR:     LD   A,(HL)
51 E02D 23                      INC  HL
52 E02E CD8D00                  CALL GRPPRT
53 E031 10F9                    DJNZ CHOSCR
54
55                   ;Onderstreping v.d. tekst
56
57                   ;CRSR op X=47, Y=27
58 E033 012F00                  LD   BC,2FH
59 E036 111B00                  LD   DE,1BH
60 E039 CD1101                  CALL MAPXYC
61
62                   ;Kleur v.d. lijn
63 E03C 3E0F                    LD   A,0FH
64 E03E CD1A01                  CALL SETATR
65
66                   ;Teken de lijn
67                   ;Voor de lengte v.d. lijn
68 E041 0698                    LD   B,98H
69                   ;Zet stip op het scherm
70 E043 CD2001      SETPIX:     CALL SETC
71                   ;CRSR plaats naar rechts
72 E046 CDFC00                  CALL RIGHTC
73 E049 10F8                    DJNZ SETPIX
74
```

Het eerste wat op het scherm verschijnt is de tekst die in de tabel op het eind van het programma is opgeslagen. De plaats van de tekst op het scherm wordt door een zogenaamde "grafische cursor" aangewezen. Voor de coördinaten van de grafische cursor zijn twee geheugenplaatsen gereserveerd, GRPACX (FCB7) voor de X- en GRPACY (FCB9) voor de Y-coördinaat. Deze worden in de regels 38 tot en met 40 van hun inhoud voorzien. Voor de kleur van de tekst wordt het kleurnummer in FORCLR (F3E9) geladen. Daarna kunnen de karakters op het scherm worden geplaatst. Daarvoor moet net zo veel keren de routine GRPPRT worden aangeroepen als er karakters op het scherm moeten komen (regels 48 tot en met 53). Voor het tekenen van lijnen is er ook een (denkbeeldige) cursor. Deze wordt met de routine MAPXYC op zijn plaats gezet. De bedoeling is de tekst te onderlijnen. Er moet daarom een horizontale lijn op het scherm worden getekend. Voor het begin van de lijn wordt de cursor op X = 47 en Y = 27 geplaatst. Deze gegevens moeten in respectievelijk BC en DE worden geladen alvorens MAPXYC kan worden aangeroepen. Voor de kleur van de lijn moet het kleurnummer met SETATR worden ingevoerd. De Accu moet het kleurnummer bevatten. Het tekenen van de lijn is verder eenvoudig. Met SETC wordt de stip aangezet op de plaats waar de cursor zich bevindt. Door de cursor voor een aantal keren naar rechts te verplaatsen met de routine RIGHTC en met SETC

steeds weer de aangewezen stip aan te zetten, ontstaat de horizontale, rechte lijn.

```
                   ;Teken een will. rechte


                   X1:        EQU   0E800H
                   Y1:        EQU   0E802H
                   X2:        EQU   0E804H
                   Y2:        EQU   0E806H
                   d2X:       EQU   0E808H
                   d2Y:       EQU   0E80AH
                   Sn:        EQU   0E80CH


                   ;Reset registers
E04B AF                       XOR   A
E04C 2100E8                   LD    HL,X1
E04F 060E                     LD    B,0EH
E051 77            RESREG:    LD    (HL),A
E052 23                       INC   HL
E053 05                       DEC   B
E054 20FB                     JR    NZ,RESREG

                   ;Invoeren v.d. co-ordinaten
E056 3E0A                     LD    A,0AH
E058 3200E8                   LD    (X1),A
E05B 3E32                     LD    A,32H
E05D 3202E8                   LD    (Y1),A
E060 3EE6                     LD    A,0E6H
E062 3204E8                   LD    (X2),A
E065 3E96                     LD    A,096H
E067 3206E8                   LD    (Y2),A
```

Het volgende onderdeel van het programma is het tekenen van een rechte lijn, schuin over het scherm. X1,Y1 en X2,Y2 zijn respectievelijk het begin- en het eindpunt van de lijn. De hoek met de X-as is kleiner dan 45 graden en de waarden dX en dY zijn beide positief. Dat betekent dat we een routine moeten maken volgens de herleiding die hiervoor gegeven is. Het is niet zo dat de volgende routine algemeen toepasbaar is. Maar voor de andere mogelijke standen die een lijn kan aannemen kan de routine worden aangepast. Deze aanpassing houdt in dat vastgesteld moet worden of dY groter is dan dX. Zo ja, dan moeten in de routine de Y en de X coördinaten worden verwisseld. Als dX negatief is moet de cursor steeds een plaats naar links worden gebracht en als dY negatief is moet de cursor steeds een plaats naar boven worden gebracht. Verder moet er voor worden gezorgd dat alle waarden die worden gebruikt, positief zijn. Bedoeld wordt dat met de absolute waarden wordt gewerkt. Het eerste deel van de routine is voor het benoemen van de diverse geheugenplaatsen. De X- en Y-waarden kunnen nooit groter zijn dan 255 (de Y-waarde zelfs niet groter dan 191). Dat zijn dus ook de grootste waarden die dX en dY kunnen aannemen. Omdat echter optellingen worden verricht zijn we genoodzaakt er rekening mee te houden dat zestienbits getallen kunnen voorkomen. Voor alle getallen zijn daarom twee geheugenplaatsen gereserveerd. Deze worden eerst gereset. Daarna worden de gegevens voor X1 (10, 0AH), Y1 (50, 32H), X2 (230, E6H) en Y2 (150, 96H) in de geheugenplaatsen gebracht. Alleen de lage byte wordt ingevoerd. De hoge byte is 0. Vandaar het resetten van de geheugenplaatsen.

```
                   ;Bepaal dX en dY

E06A 0602                     LD    B,02H
                   ;1e doorgang, dX
E06C DD2100E8                 LD    IX,X1
E070 DD7E04        DISCR:     LD    A,(IX+4)
E073 DD9600                   SUB   (IX+0)
E076 DD7708                   LD    (IX+8),A
E079 DD23                     INC   IX
E07B DD23                     INC   IX
                   ;2e doorgang, dY
E07D 10F1                     DJNZ  DISCR


                   ;Bepaal aantal stappen
E07F 2A08E8                   LD    HL,(d2X)
E082 EB                       EX    DE,HL
E083 2A0AE8                   LD    HL,(d2Y)
E086 19                       ADD   HL,DE
E087 220CE8                   LD    (Sn),HL

                   ;Bereken 2*dY
E08A 2A0AE8                   LD    HL,(d2Y)
E08D 29                       ADD   HL,HL
E08E 220AE8                   LD    (d2Y),HL

                   ;Bereken SY
E091 A7                       AND   A
E092 ED5B08E8                 LD    DE,(d2X)
E096 ED52                     SBC   HL,DE
E098 E5                       PUSH  HL

                   ;Bereken 2*dX
E099 EB                       EX    DE,HL
E09A 29                       ADD   HL,HL
E09B 2208E8                   LD    (d2X),HL
```

Het volgende deel is het initiëren van de routine. Eerst worden dX en dY berekend. dX uit X2 − X1 en dY uit Y2 − Y1. Voor de verandering is hiervoor de geïndexeerde adresseermethode gevolgd. In IX wordt het adres van X1 geladen. Door hier 4 bij op te tellen (IX + 4) wordt het adres van X2 gevormd. De waarde hiervan wordt in de Accu geladen. Hiervan wordt de inhoud van X1 afgetrokken: SUB (IX + 0). Nu is de waarde dX gevonden die in d2X wordt geladen: LD (IX + 8),A. Door twee keer INC IX vinden we het adres van Y1 in het IX register. In de tweede lus herhaalt zich alles voor dY, waarvan we de waarde in d2Y vinden. Voor het aantal af te leggen stappen moeten dX en dY worden opgeteld. Omdat hieruit een getal kan ont-

staan dat groter is dan acht bits wordt de berekening zestienbits uitgevoerd (regels 120 tot en met 124). Het berekenen van 2dY betekent niets meer dan het optellen van dY bij zichzelf. Ook deze berekening moet voor een zestienbits getal worden uitgevoerd. Uiteindelijk bevat d2Y het resultaat. Met het berekenen van 2dX wachten we nog even. Eerst moet SY = 2dY − dX worden berekend. De waarde 2dY bevindt zich nog in HL.
Hiervan wordt dX afgetrokken door dX in DE te laden en SBC HL,DE uit te laten voeren. De waarde voor SY wordt in de STACK bewaard. Hierna is de beurt aan het berekenen van 2dX.

```
142                        ;Cursor op X1,Y1
143 E09E ED4B00E8                 LD   BC,(X1)
144 E0A2 ED5B02E8                 LD   DE,(Y1)
145 E0A6 CD1101                   CALL MAPXYC
146
147                        ;Teken de lijn
148 E0A9 ED4B0CE8                 LD   BC,(Sn)
149                        ;SY in HL
150 E0AD E1                       POP  HL
151 E0AE CD2001   DRWLIN:         CALL SETC
152                        ;Een plaats naar rechts
153 E0B1 CDFC00   XJMP:           CALL RIGHTC
154 E0B4 0B                       DEC  BC
155                        ;Bepaal teken van SY
156 E0B5 7C                       LD   A,H
157 E0B6 A7                       AND  A
158 E0B7 FAC5E0                   JP   M,YJMP1
159                        ;Een plaats naar beneden
160 E0BA CD0801   YJMP:           CALL DOWNC
161 E0BD 0B                       DEC  BC
162                        ;Bereken SY-2dX
163 E0BE A7                       AND  A
164 E0BF ED5B08E8                 LD   DE,(d2X)
165 E0C3 ED52                     SBC  HL,DE
166                        ;Bereken SY+2dY
167 E0C5 ED5B0AE8 YJMP1:          LD   DE,(d2Y)
168 E0C9 19                       ADD  HL,DE
169                        ;Bepaal of alle stappen
170                        ;zijn gemaakt
171 E0CA 78                       LD   A,B
172 E0CB B1                       OR   C
173                        ;Terug voor volgend punt
174 E0CC 20E0                     JR   NZ,DRWLIN
175
```

Nu begint de eigenlijke routine voor het tekenen van de lijn. Het eerste wordt het punt X1,Y1 getekend. Daarvoor moet de cursor naar deze plaats op het scherm worden gebracht met MAPXYC. Het aantal stappen wordt in BC geplaatst. Steeds als de cursor een stap is verplaatst wordt BC met 1 verminderd. Als alle stappen zijn uitgevoerd is de inhoud van BC 0. Het volgende is dat met POP HL, SY in HL wordt geladen. Nu wordt de eerste stip aangezet met SETC. Dit is de plaats waar we steeds weer in de routine naar terugkeren. Het verplaatsen van de cursor naar rechts is bij deze lijn in elk geval nodig (verlaag BC). Of ook een stap in Y-richting moet worden gedaan is afhankelijk van het teken van SY. Dit kan worden bepaald door de hoge byte van SY (in H) te onderzoeken. Daarvoor wordt H in A geladen en met AND A wordt aan de S-FLAG de juiste waarde gegeven. Bij een negatieve waarde van SY mag geen stap in Y-richting worden uitgevoerd en worden de regels 160 tot en met 165 overgeslagen. Bij een positieve waarde van SY wordt door CALL DOWNC de cursor een plaats naar onderen gebracht. Aansluitend hierop wordt BC weer met 1 verminderd. Als een stap in Y-richting is gemaakt dan moet SY met 2dX worden verminderd. De regels 163 tot en met 165 verzorgen dat. Er is in elk geval een stap in X-richting gemaakt. Er moet daarom altijd bij SY de waarde 2dY worden opgeteld. Dat wordt verricht door de regels 167 en 168, die ook worden doorlopen als er geen stap in Y-richting is gemaakt. Het geheel wordt herhaald tot alle stappen zijn gemaakt.

```
176                     ;Wacht op CTRL-STOP
177 E0CE CDB700         BREAK:      CALL BREAKX
178 E0D1 3802                       JR   C,RSTSCR
179 E0D3 18F9                       JR   BREAK
180
181                     ;Terug naar schermmode 0
182 E0D5 AF             RSTSCR:     XOR  A
183 E0D6 CD5F00                     CALL CHGMOD
184 E0D9 C9                         RET
185
186 E0DA 47726166 TABEL:            DB   'Graf'
187 E0DE 69736368                   DB   'isch'
188 E0E2 65206675                   DB   'e fu'
189 E0E6 6E637469                   DB   'ncti'
190 E0EA 65732E                     DB   'es.'
191                                 END
```

De laatste regels van het programma kunnen u, dacht ik, nu geen moeilijkheden meer geven.

12.3. De multicolormode.

De multicolormode heeft veel overeenstemming met de hoge resolutiemode, hoewel er van hoge resolutie bepaald niet meer kan worden gesproken. Ook in de multicolormode worden er stippen op het scherm geplaatst, maar die zijn in de hoogte en in de breedte maar liefst vier keer zo groot als de stippen waaruit het veld is opgebouwd. We zullen daarom maar niet meer van stippen spreken maar van "blokjes" en elk blokje beslaat een rooster van vier bij vier stippen. Ook bij deze mode is er sprake van een bitmapgeheugen (de oorspronkelijke karaktergenerator) en een schermgeheugen. Dit laatste is weer gevuld met oplopende getallen. De indeling van het bitmapgeheugen komt weer overeen met dat van de hoge resolutiemode. De eerste

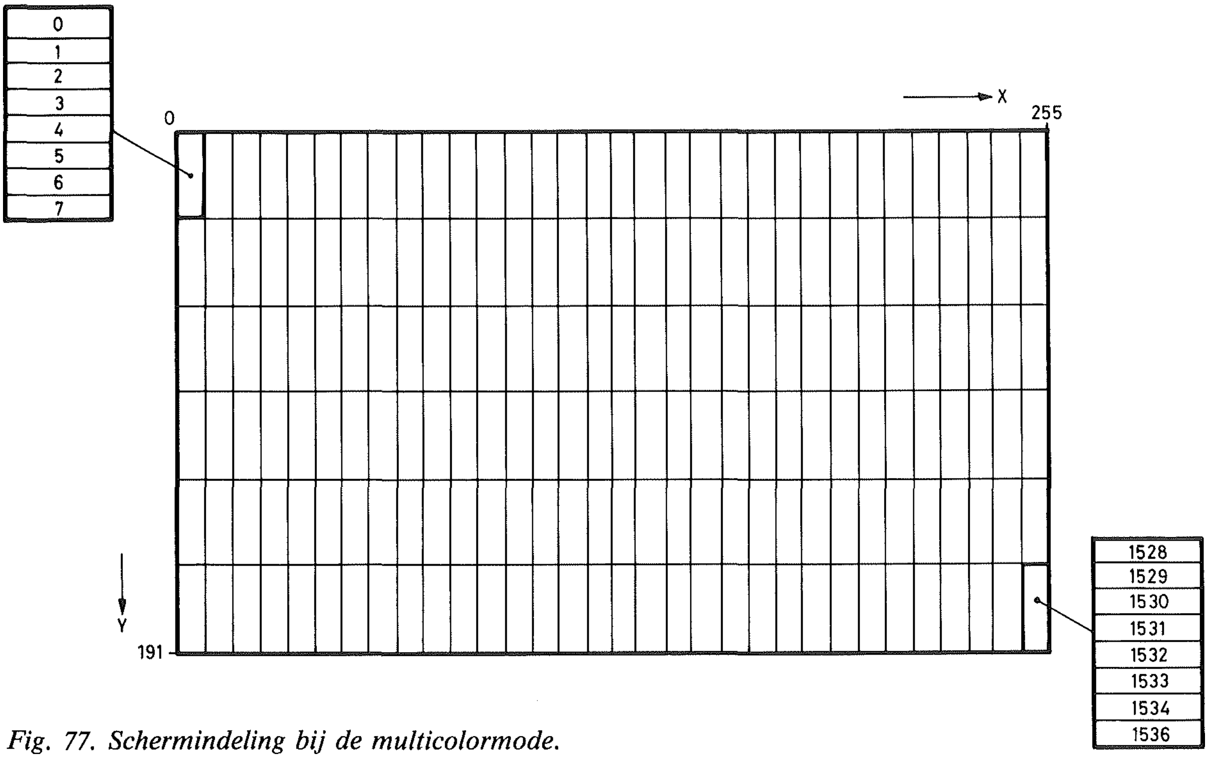


Fig. 77. Schermindeling bij de multicolormode.

acht geheugenplaatsen corresponderen met een rooster van twee blokjes breed en acht blokjes hoog. Elke geheugenplaats bevat de gegevens voor twee naast elkaar liggende blokjes. De volgende acht geheugenplaatsen bevatten weer de gegevens voor het daarnaast liggende rooster van twee blokjes breed en acht blokjes hoog. In de hoogte van het scherm passen zo zes van deze roosters (aantal stippen: $6*8*4=192$). In totaal passen $6*32=192$ roosters op het scherm. Voor elk rooster zijn acht geheugenplaatsen nodig zodat het bitmapgeheugen er $8*192=1536$ moet bevatten. Figuur 77 geeft een voorstelling van de indeling van het scherm. Het bitmapgeheugen is tevens het kleurgeheugen. De lage tetrade van een geheugenplaats bevat het nummer van de kleur die het rechterblokje moet aannemen en de hoge tetrade het nummer van de kleur die het linkerblokje moet aannemen.

13. ROM-routines

13.1. Algemeen.

In dit hoofdstuk worden een aantal ROM-routines opgesomd die kunnen worden aangeroepen vanuit adressen van het BASIC Input/Output System (BIOS). Op deze adressen vindt men een JP-instructie met het adres waar de desbetreffende routine in de BASIC-ROM in werkelijkheid aanvangt. Het is natuurlijk mogelijk voor een bepaalde computer een lijst samen te stellen van deze adressen. Het gebruiken van de BIOS ''entry points'' garandeert echter dat het programma op elke MSX-computer dezelfde werking zal hebben. Een aantal van deze routines zijn al gebruikt in de programma's die hiervoor zijn gegeven. Deze toepassingen maakten het mogelijk van de desbetreffende routines de werking duidelijk te maken. Waar nodig worden in dit hoofdstuk ook enige aanwijzingen gegeven betreffende het gebruik van de routines. Bij de volgende opsomming wordt eerst de naam van de routine gegeven met direct daarna het adres (hexadecimaal) waarop deze is aan te roepen. Dit aanroepen kan in een enkel geval met een RST X instructie. In alle gevallen kunnen de routines worden aangeroepen met een CALL- instructie naar het opgegeven BIOS adres of vanuit BASIC met het DEFUSR-statement. Verder wordt ook in het kort de werking van de routines gegeven en de eventueel gebruikte geheugenplaatsen in de systeem-RAM. Als vermeld is dat onder een zekere voorwaarde een FLAG een bepaalde waarde krijgt, dan zal deze FLAG de tegengestelde waarde hebben als niet aan de voorwaarde is voldaan.

13.2. Restart-routines.

CHKRAM -0000-
Na het inschakelen van de computer wordt als eerste deze routine gestart. De routine kan alleen worden gebruikt voor het resetten van de computer.

SYNCHR -0008-
De BASIC-interpreter gebruikt deze routine om na te gaan of een gelezen karakter in overeenstemming is met reeds eerder gelezen karakters. Zo niet, dan volgt ''Sintax error''.

CHRGTR -0010-
Wordt gebruikt door de BASIC-interpreter voor het lezen (een FETCH-handeling) van het volgende karakter van het BASIC- programma.

DCOMPR -0020-
Deze routine vergelijkt de inhouden van HL en DE. Als HL = DE, dan wordt de Z-FLAG geset en als HL kleiner is dan DE, wordt de Carry-FLAG geset.

KEYINT -0038-
De MSX-computer werkt met de interruptmode 1. Elke 20 ms. wordt een interrupt veroorzaakt. Hierdoor wordt deze routine aangeroepen. Behalve het verhogen van de Jiffy clock is een van de belangrijke functies van de routine het scannen van het toetsenbord. De inhoud van geen enkel processorregister wordt veranderd. Als een toets is ingedrukt wordt het codegetal uit de MSX-karakterset van het desbetreffende karakter in de Keyboard buffer geplaatst. De HOOK KEYI wordt aangeroepen.

13.3. Routines voor het toetsenbord, scherm en printer.

CHGET -009F-
Deze routine leest een karakter uit de Keyboard buffer en wacht eventueel op het indrukken van een toets. Behalve de Accu, die het codegetal van het desbetreffende karakter bevat, worden geen registerinhouden veranderd.
De HOOK CHGE wordt aangeroepen.

CHPUT -00A2-
Het karakter, waarvan het codegetal zich in de Accu bevindt, wordt op het scherm geprint. Voor de plaats van de cursor kan de routine POSIT worden gebruikt. Processorregisters worden niet veranderd.

De HOOK CHPU wordt aangeroepen.

LPTOUT -00A5-
Een karakter waarvan het codegetal zich in de Accu bevindt wordt naar de printer verzonden. Indien de printer niet gereed is om een karakter te ontvangen, dan wacht de routine. Processorregisters worden niet veranderd. Als het karakter is geprint wordt de Carry-FLAG geset. Deze wordt gereset als de stoptoets is gebruikt.
De HOOK LPTO wordt aangeroepen.

INLIN -00B1-
De BASIC-interpreter gebruikt de routine voor het ontvangen van een regel van het Keyboard. Hij print de karakters op het scherm en plaatst ze in een buffer. Het eind van de regel is het gebruiken van de RETURN of de CTRL-STOP toetsen. Na het verwerken van de routine geeft HL het adres-1 van de top van de buffer.
De registers AF, BC, DE en HL kunnen zijn veranderd.

BREAKX -00B7-
Controleert de CTRL-STOP-toetsen. Zijn deze gebruikt dan is de Carry FLAG geset. Alleen de inhoud van de Accu wordt veranderd. Kan ook worden gebruikt als geen interrupt mogelijk is.

ISCNTC -00BA-
Bepaalt of de CTRL-STOP-toetsen zijn ingedrukt of wanneer alleen de STOP-toets is gebruikt. Wordt alleen de STOP-toets gebruikt dan stopt het programma tot opnieuw deze toets is ingedrukt.

Worden de CTRL-STOP-toetsen ingedrukt dan wordt teruggegaan naar de BASIC-commando-mode. De geheugenplaats BASROM (FBB1) moet "0" zijn (is bij het inschakelen van de computer gereset). Is BASROM niet 0 dan heeft de routine geen werking. Alleen de inhoud van de Accu kan zijn veranderd.

BEEP -00C0-
Deze routine verzorgt het bekende BEEP-geluid. De registers AF, BC, DE en HL kunnen zijn veranderd.

CLS -00C3-
De bedoeling is dat hierdoor het scherm (in *elke* mode) wordt "schoon" gemaakt. Dat kan alleen als de Z-FLAG is geset.
De registers AF, BC en DE kunnen zijn veranderd.

POSIT -00C6-
Hiermee wordt de cursor in de TEKST-mode op een bepaalde positie gebracht. Het kolomnummer wordt in H en het regelnummer in L geladen. CSRX (F3DD) en CSRY (F3DC) bewaren het cursoradres. Register AF kan zijn veranderd.

FNKSB -00C9-
Deze routine gaat na of de inhoud van geheugenplaats CNSDFG (F3DE) nul is. In dat geval heeft hij geen werking. Is CNSDFG niet nul dan wordt de routine DSPFNK aangeroepen voor het printen van het functietoetsendisplay. De registers AF, BC en DE kunnen zijn veranderd.

ERAFNK -00CC-
Als in een tekstmode het functietoetsendisplay "aan" staat dan wordt het door deze routine van het scherm verwijderd. De registers AF, BC en DE kunnen zijn veranderd.

DSPFNK -00CF-
Print het functietoetsendisplay op het scherm. De registers AF, BC en DE kunnen zijn veranderd. De inhoud van CSNDFG wordt FFH (zie FNKSB).

TOTEXT -00D2-
De routine brengt het scherm in de tekstmode die door de inhoud van OLDSCR (FCB0) wordt aangegeven. Hij roept de HOOK TOTE aan met de inhoud van OLDSCR in de Accu. De processorregisters AF, BC, DE en HL kunnen zijn veranderd.

SNSMAT -0141-
Zie voor het gebruik van deze routine paragraaf 9.5. Alleen het register AF kan zijn veranderd.

13.4. Routines voor het gebruik van de tape.

TAPION -00E1-
Zet de cassettemotor aan en leest de "HEADER". Bepaalt door middel hiervan de BAUD-snelheid. Veranderd worden de registers AF, BC, DE en HL. Aan het eind wordt de routine BREAKX aangeroepen.

TAPIN -00E4-
De routine leest een karakter van de tape. De Accu bevat het gelezen karakter. De routine BREAKX wordt gebruikt voor het lezen van de CTRL-

STOP-toetsen. De registers AF, BC, DE en HL worden veranderd.

TAPIOF -00E7-
Einde van het lezen van de tape, de cassettemotor wordt gestopt. Er worden geen processorregisters veranderd.

Bij elke file die naar de tape wordt geschreven zijn twee delen te herkennen, de LABEL, waarin de naam van de file voorkomt en daarna de file zelf. Voor het lezen van de LABEL worden eerst de drie bovenstaande routines gebruikt (inclusief TAPIOF). Dan wordt nagegaan of de juiste LABEL is ontvangen en zo ja, dan wordt de file zelf gelezen. Hierbij moeten weer de drie bovenstaande routines worden gebruikt, inclusief TAPION.

TAPOON -00EA-
Zet de cassettemotor aan en schrijft de HEADER naar de tape. Voor een lange HEADER moet de Accu een getal bevatten, niet 0. Voor een korte HEADER moet de Accu "0" zijn. De registers AF, BC, DE en HL worden veranderd. De BREAKX-routine wordt gebruikt voor het controleren van de CTRL-STOP-toetsen.

TAPOUT -00ED-
Hiermee wordt een karakter (een byte) naar de tape geschreven. De Accu moet het codegetal van het karakter bevatten. De registers AF, BC, DE en HL worden veranderd. De routine gebruikt BREAKX.

TAPOOF -00F0-
Het schrijven naar de cassette wordt gestopt. De cassettemotor wordt uitgezet. Er worden geen processorregisters veranderd.

Met de BAUD-snelheid wordt bedoeld het aantal bits dat per seconde wordt verzonden. De bits worden "in serie" naar de tape gezonden en voor elk karakter zijn er 11 nodig. Bij een BAUD-snelheid van 2400 b/s kunnen dus in een seconde 2400/11 = 218 karakters worden verzonden. Het verschil tussen een "1" en een "0" is merkbaar aan de frequentie van de verzonden toon. Een "hoge" frequentie is "1" en een "lage" frequentie is "0". Bij 1200 BAUD: "0" = 1200 Hz en "1" = 2400 Hz. Bij 2400 BAUD: "0" = 2400 Hz en "1" = 4800 Hz. Om de computer, bij het lezen van de tape, gelegenheid te geven de BAUD-snelheid te bepalen worden in de LABEL de karakters van de naam voorafgegaan door een "lange HEADER". Dit is een toon, gedurende een bepaalde tijd (6,67 s.), van een frequentie die overeenkomt met een "1". De file zelf wordt voorafgegaan door een "korte HEADER" (1,67 s.). Voor het instellen van de BAUD-snelheid moeten een aantal geheugenplaatsen van een getal worden voorzien. Deze plaatsen zijn LOW, F406, 2 byte; HIGH, F408, 2 byte en HEADER, F40A, 1 byte.

Voor 1200 BAUD:
F406: 54H
F407: 5CH
F408: 26H
F409: 2DH
F40A: 0FH

Voor 2400 BAUD:
F406: 25H
F407: 2DH
F408: 0EH
F409: 16H
F40A: 1FH

13.5. Routines voor het gebruik van de PSG.

GICINI -0090-
Voor het initiëren van de Programmable Sound Generator. De diverse registers worden gereset. Er worden geen processorregisters veranderd.

WRTPSG -0093-
De data die is geplaatst in register E wordt geschreven in het PSG-register waarvan het nummer is gegeven door de Accu. Er worden geen processorregisters veranderd.

RDPSG -0096-
Het PSG-register waarvan het nummer in de Accu is opgeslagen, wordt gelezen. De waarde in dat register wordt na het doorlopen van de routine in de Accu gevonden. Geen verdere processorregisters worden veranderd.

13.6. Routines voor het gebruik van de VDP.

DISSCR -0041-
Disable screen, maakt het scherm blank. De processorregisters AF en BC worden veranderd.

ENASCR -0044-
Enable screen, laat de inhoud van het scherm weer zien. De registers AF en BC worden veranderd.

WRTVDP -0047-
De write only registers van de Video Display Processor kunnen met deze routine worden ingeschreven. Het registernummer moet daarvoor in C worden geladen en de data die voor dat register bestemd is moet in B worden geschreven. Dezelfde waarde die in de VDP-write only registers is geschreven bevindt zich ook in de geheugenplaatsen RGxSAV, waarin x gelijk is aan het desbetreffende registernummer. De volgende geheugenplaatsen worden hiervoor gebruikt:
RG0SAV: F3DF
RG1SAV: F3E0
RG2SAV: F3E1
RG3SAV: F3E2
RG4SAV: F3E3
RG5SAV: F3E4
RG6SAV: F3E5
RG7SAV: F3E6

RDVDP -013E-
Deze routine wordt gebruikt voor het lezen van het VDP-statusregister VDP-8. Dit is een read only register en de inhoud van dit register komt in de Accu. Er worden verder geen processorregisters veranderd.

RDVRM -004A-
Hiermee kan een register van de VideoRAM worden gelezen. Het adres van dat register moet in HL worden geschreven. De inhoud van het geadresseerde register vinden we terug in de Accu. Alleen de Accu krijgt bij deze routine een andere waarde.

WRTVDM -004D-
De waarde in de Accu wordt geschreven in het register van de VideoRAM waarvan het adres gegeven is door HL. Alleen het AF- registerpaar wordt veranderd.

FILVRM -0056-
Deze routine vult een blok van de VideoRAM met eenzelfde waarde. Het beginadres van het blok is gegeven door HL, de lengte van het blok door BC en de waarde waarmee de geheugenplaatsen moeten worden gevuld, door de Accu. De registers AF en BC worden veranderd.

LDIRMV -0059-
Een geheugenblok in de VideoRAM, geadresseerd door HL en waarvan de lengte door BC is gegeven, wordt overgebracht naar het "normale" geheugen op het adres dat gegeven is door DE. AF, BC en DE zijn de registers die worden veranderd.

LDIRVM -005C-
Een blok geheugenplaatsen uit het "normale" geheugen en geadresseerd door HL, wordt overgebracht naar de VideoRAM. De lengte van het blok is in BC gegeven en het doeladres in DE. De registers AF, BC en DE veranderen.

CHGMOD -005F-
Met deze routine kan de schermmode worden gekozen. Het nummer van de desbetreffende mode moet in de Accu worden geladen. De registers AF, BC, DE en HL worden veranderd. De geheugenplaats SCRMOD (FCAF) van het systeem RAM bevat het nummer van de mode waarin het scherm zich bevindt.

CHGCLR -0062-
Bij deze routine wordt de inhoud van de volgende drie geheugenplaatsen gelezen. Deze moeten de kleurnummers voor het scherm bevatten.
FORCLR (F3E9); voorgrondkleur.
BAKCLR (F3EA); achtergrondkleur.
BDRCLR (F3EB); kleur van het kader.
Bij mode 0 wordt FORCLR en BAKCLR gelezen voor de voorgrondkleur van het veld en de achtergrondkleur van zowel het veld als het kader. Bij mode 1 doen alle drie de geheugenplaatsen dienst. Bij mode 2 en 3 slechts alleen geheugenplaats BDRCLR. De inhoud van de genoemde geheugenplaatsen verandert niet. Wel de registers AF, BC en HL.

CLRSPR -0069-
Deze routine initieert de spriteregisters. De sprites worden onzichtbaar. Alle spritepatronen worden met "0" ingeschreven. In het kenmerkgeheugen wordt de Y-waarde 209, De kleur wordt gelijk aan de achtergrondkleur van het veld en het patroonnummer wordt gelijk aan het spritenummer. De registers AF, BC, DE en HL veranderen.

CALPAT -0084-
De routine geeft in HL het patroonadres van de sprite waarvan het nummer in de Accu moet zijn

geladen. De registers AF, DE en HL veranderen door het aanroepen van deze routine.

GRPPRT -008D-
Deze routine print het karakter, waarvan het codegetal in de Accu is geladen, op het grafische scherm. De plaats van het karakter wordt aangegeven door de "grafische cursor" waarvan de X- en de Y-waarde zijn gegeven in de geheugenplaatsen GRPACX, 2 byte (FCB7), en GRPACY, 2 byte (FCB9). De kleur van de karakters komt overeen met de kleur die in FORCLR (F3E9) is gegeven. Er worden geen processorregisters veranderd.

MAPXYC -0111-
Voor een gegeven X- en Y-waarde van een stip op het grafische scherm, moet een lastige berekening worden gemaakt om het adres te vinden in het bitmapgeheugen van het byte, waarin een bit de plaats van de stip vertegenwoordigt. Deze berekening wordt uitgevoerd door MAPXYC. De X-waarde moet hiervoor in BC en de Y- waarde in DE. Het adres in het bitmapgeheugen van het desbetreffende byte wordt gegeven in geheugenplaats CLOC (F92A), 2 byte, en de plaats van het bit in CMASK (F92C), 1 byte. In CMASC bevindt zich een binair getal (het masker) waarvan slechts een enkel bit 1 is. De plaats hiervan is tevens de plaats van het gevraagde bit. Bij kruisende lijnen op het scherm kan het zijn dat in een byte van het bitmapgeheugen meerdere bits 1 moeten zijn. Met deze mogelijkheid kan worden gerekend door de volgende instructies toe te passen (na het aanroepen van MAPXYC):

```
LD HL,(CLOC)
CALL RDVRM
LD HL,CMASK
OR (HL)
LD (HL),A
```

De routines die betrekking hebben op het tekenen van lijnen op het scherm gebruiken allemaal de geheugenplaatsen CLOC en CMASC, zodat uitlezen van deze plaatsen in de meeste gevallen niet nodig zal zijn. De gegevens in CLOC en CMASC worden ook wel de *"grafische cursor"* genoemd. Door MAPXYC worden de registers AF, D en HL veranderd.

FETCHC -0114-
De werking van deze routine komt overeen met
LD A,(CMASK)
LD HL,(CLOC)
De andere processorregisters worden niet veranderd.

STOREC -0117-
De werking van deze routine komt overeen met
LD (CMASK),A
LD (CLOC),HL
De andere processorregisters worden niet veranderd.

SETATR -011A-
Bij het tekenen van lijnen op het scherm wordt voor de kleur gekeken naar de geheugenplaats ATRBYT (F3F2) waarin zich het kleurnummer bevindt (attribute byte). Het invoeren van het kleurnummer wordt door SETATR verricht. Hiervoor moet de Accu het kleurnummer bevatten. Er veranderen geen processorregisters.

SETC -0120-
Door deze routine verkrijgt de stip, die geadresseerd wordt door CLOC en CMASC, de kleur die wordt gegeven door ATRBYT. Register AF wordt veranderd.

RIGHTC -00FC-
De grafische cursor gaat een plaats naar rechts. AF wordt veranderd.

LEFTC -00FF-
De grafische cursor gaat een plaats naar links. AF wordt veranderd.

UPC -0102-
De grafische cursor gaat een plaats omhoog, tenzij de bovenkant van het veld reeds is bereikt. AF wordt veranderd.

DOWNC -0108-
De grafische cursor gaat een plaats naar beneden, tenzij de cursor de onderkant van het veld reeds heeft bereikt. AF wordt veranderd.